



**LABSOFT™
REFERENCE
MANUAL**

Cyborg®



LABSOFT REFERENCE MANUAL

VERSION 1.3

04/15/83

PART # 821-049

*
*
* LABSOFT REFERENCE MANUAL *
*
*
*

Cyborg Part #821-049
Version 1.3 04/15/83

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, broadcasting, recording, photocopying, or by any means or process not yet in use, for any purpose whatsoever, without the express written permission of Cyborg Corporation.

Cyborg Corporation makes no warranty of any kind with regard to this material, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Cyborg Corporation shall not be held liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual.

Written by R. O. Curtis
(with R. M. Mottola and Jerry Reilly)

Copyright © 1983 by Cyborg Corporation
Cyborg, ISAAC, and LabSoft are trademarks of Cyborg Corp.
Apple and Applesoft are trademarks of Apple Computer Co.

Cyborg Corp., 55 Chapel St., Newton, MA 02158 (617) 964-9020

THE LABSOFT REFERENCE MANUAL

TABLE OF CONTENTS

CHAPTER 1:	INTRODUCTION ...	1
1.1	The ISAAC Reference Library ...	1
1.2	Syntax Conventions Used in This Manual ...	3
CHAPTER 2:	THE LABSOFT MASTER DISK ...	5
2.1	Bootting The Disk ...	5
2.2	Where LabSoft Will Live ...	6
2.3	Bootting LabSoft From Your Own Programs ...	6
2.4	LabSoft's Relationship to Applesoft ...	7
2.5	The LabSoft Utilities Disk ...	8
CHAPTER 3:	THE UTILITY COMMANDS ...	9
3.1	The Beeper/Buzzer Commands ...	9
3.2	& PAUSE ...	10
3.3	& ARRAYCLR ...	10
3.4	& OFFERR ...	11
3.5	& ERRPTCH ...	11
3.6	& SLOT# ...	11
3.7	& FMTDFLT ...	12
CHAPTER 4:	THE GRAPHING COMMANDS ...	13
	The Scrolling Graph ...	13
	The Alternating Graph ...	15
	Graphing Analog Values ...	16
	Graphing Binary Values ...	17
	A Note on Memory Management ...	18
	Graphing Commands and Graphing Routines ...	18
4.1	& HIRES1 ...	19
4.2	& HIRES2 ...	19
4.3	& FULLSCREEN ...	19
4.4	& MIXSCREEN ...	20
4.5	& SCROLLSET ...	20
4.6	& ALTSET ...	21
4.7	& OUTLINE ...	21
4.8	& RETRCE ...	22
4.9	& PLTFMT ...	22
4.10	& NXTPLT ...	24
4.11	& NXTBIN ...	24
4.12	& BINFMT ...	25
4.13	& LABEL ...	25
	Analog Graphing Routines ...	28
	Binary Graphing Routines ...	30

CHAPTER 5:	THE INPUT/OUTPUT PARAMETERS ...	33
	Structure of the I/O Commands and Parameters ...	33
	5.1 (TV) ...	34
	5.2 (DV) ...	34
	5.3 (AV) ...	35
	About Data Arrays ...	35
	5.4 (TH) ...	37
	5.5 (D#) ...	37
	5.6 (C#) ...	38
	5.7 (FU) ...	39
	Using (FU) With Input Commands ...	39
	Using (FU) With Output Commands ...	41
	5.8 (XM) ...	42
	5.9 (AM) ...	42
	Using "AND" Masks ...	42
	Using "XOR" Masks ...	43
	5.10 (GA) ...	45
	5.11 (GB) ...	46
	5.12 (PR) ...	47
	5.13 (RT) ...	47
	5.14 (SW) ...	48
	5.15 (CV) ...	48
	5.16 (W#) ...	49
	A Note on Matrix and Triggered Operations ...	50
	I/O Sequences and Reserved Variables ...	52
CHAPTER 6:	THE ANALOG I/O COMMANDS ...	55
	6.1 & AIN	
	& @AIN ...	56
	6.2 & ASUM	
	& @ASUM ...	57
	Analog Signal Averaging Using & ASUM ...	57
	6.3 & AOUT	
	& @AOUT ...	58
	6.4 & ANAFMT ...	60
	How Sampled Data is Stored in an Array ...	61
CHAPTER 7:	THE BINARY I/O COMMANDS; THE SCHMITT TRIGGER COMMANDS ...	63
	7.1 & BIN	
	& @BIN ...	64
	Binary Input Handshaking ...	65
	7.2 & BOUT	
	& @BOUT ...	67
	Binary Output Handshaking ...	68
	7.3 & BCDIN	
	& @BCDIN ...	70
	How LabSoft Interprets BCD ...	71
	7.4 & BCDOUT	
	& @BCDOUT ...	72

7.5	& BPOLL	
	& @BPOLL ...	73
	The Schmitt Trigger Commands ...	75
7.6	& TRIGIN	
	& @TRIGIN ...	75
	How LabSoft Reads Schmitt Trigger States ...	76
7.7	& TRIGPOLL	
	& @TRIGPOLL ...	77
CHAPTER 8:	THE COUNTER COMMANDS ...	79
8.1	& CLRCOUNTER ...	80
8.2	& CNTMT ...	80
	Counter Channel Formatting ...	81
8.3	& COUNTERIN ...	82
8.4	& FINL	
	& @FINL ...	83
	Software Frequency Measurement ...	84
8.5	& FINH	
	& @FINH ...	85
CHAPTER 9:	THE TIMER AND CLOCK COMMANDS ...	87
9.1	& CLRTIMER ...	88
9.2	& TIMERIN ...	88
9.3	& TIME TO ...	89
9.4	& DAY TO ...	89
CHAPTER 10:	& LOOK FOR...THEN ...	91
10.1	Looking For an Input ...	92
10.2	Looking For an Output ...	95
A	Final Word ...	96
APPENDIX A:	ERROR MESSAGES GENERATED BY LABSOFT	
APPENDIX B:	LABSOFT RESERVED VARIABLES	
APPENDIX C:	LABSOFT'S OBJECT FILES	
APPENDIX D:	MEMORY MAPS	
APPENDIX E:	SOME USEFUL PEEKS, POKES AND CALLS	
APPENDIX F:	SPEEDING UP LABSOFT PROGRAMS	
APPENDIX G:	MAKING EFFICIENT USE OF MEMORY	
APPENDIX H:	LABSOFT BENCHMARKS	
APPENDIX I:	GENERATING YOUR OWN CHARACTER SET	
APPENDIX J:	BIT MASKING	
APPENDIX K:	A LABSOFT DICTIONARY	
APPENDIX L:	SOME USEFUL SUBROUTINES	
APPENDIX M:	OPTIMIZING THE USE OF INTEGER VARIABLES	
APPENDIX U:	UTILITY DISK PRINT-OUT	
APPENDIX X:	THE EXPANSION COMMANDS	
APPENDIX Z:	INDEX	

ISAAC

CHAPTER 1 INTRODUCTION

- 1.1 The ISAAC Reference Library
- 1.2 Syntax Conventions Used in This Manual



CHAPTER 1: INTRODUCTION

LabSoft is Cyborg's extension of Applesoft, the resident high-level language of the Apple II computer. If you are familiar with Applesoft (and you should be if you want to fully understand this manual), then you know that it is an extended form of the well-known BASIC programming language. BASIC, versatile as it is, can't really accomodate the many capabilities of the Apple II (color graphics, game paddles, and the speaker, to name a few), nor can Applesoft deal effectively with the many data acquisition and control capabilities of an ISAAC system. When the LabSoft instruction set is added to the already extensive Applesoft instruction set, however, the ISAAC/LabSoft/Apple system becomes a powerful tool with a wide range of data acquisition and control applications.

Before reading this manual, you should have a working knowledge of Applesoft. Although it is possible in theory to write a complete program using LabSoft alone, the vast majority of the programs written for ISAAC systems combine LabSoft and Applesoft instructions. And since the structure of these programs will be largely determined by the nature of Applesoft, we recommend that you be able to write and run simple Applesoft programs before you continue with LabSoft.

To learn about Applesoft and Apples in general, read the Applesoft Tutorial. To learn more, read the Applesoft Reference Manual. These are both terrific books; they're educational, informative, and not at all boring. We read them all the time around here.

1.1 The ISAAC Reference Library

As we mentioned, there are several volumes of documentation associated with the Apple II and several more associated with ISAAC. It may look like a great deal of reading material, but don't

let it scare you. You won't have to read it all at once, and some of it you may not have to read at all. ISAAC, like the Apple, is a general-purpose machine, which means that we have to provide information that will be of use to beginners as well as experts.

Wherever possible, we have tried to arrange this information in order of ascending complexity and index it so that you won't have to wade through information you don't need in order to find the information you do need.

The standard ISAAC Reference Library consists of three volumes. The LabSoft Reference Manual, (this book) deals exclusively with LabSoft: its commands, parameters, structure, and syntax. The ISAAC System Reference is devoted to the hardware portion of your ISAAC system: how to set it up, reconfigure it to suit your needs, and connect it with other hardware. The ISAAC User's Guide is an introduction to both ISAAC and LabSoft. The User's Guide begins with the information you need to unpack and set up the system. It continues as an interactive tutorial which will get you started using ISAAC and LabSoft to perform simple data acquisition and control operations. And it ends with a System Index, which will help you locate the information you need, no matter where it is in the ISAAC Reference Library. We think that the User's Guide is the best place to start learning about ISAAC and LabSoft.

The LabSoft Reference Manual (this book) will deal with Labsoft in detail. There are chapters covering each of the seven categories of commands, as well as separate chapters on the I/O parameters and & LOOK FOR...THEN... (triggered) command structure. There are also a number of appendices containing information that will help you to utilize LabSoft in the most efficient manner. It's a very thorough manual. That's why it's so long. You may want to read just the introductory material and the sections on the LabSoft instructions that most interest you, or you may want to read the whole thing. Whatever your choice, keep in mind that LabSoft is an applications-oriented language, and that you are the best judge of the requirements of your par-

ticular application.

You'll also want to take at least a brief look at the ISAAC System Reference. In it, you will find additional information about ISAAC's hardware devices, as well as the kind of detailed technical specifications you'll need if you plan to take advantage of ISAAC's many user-configurable hardware options.

1.2 Syntax Conventions Used in This Manual

Since we're assuming that you are familiar with Applesoft and the Applesoft Reference Manual, we have tried to follow the syntax conventions (those labels required to define allowable arguments, values, etc.) used in that manual. Those most commonly used are listed below. The rest are listed in the Applesoft Reference Manual (under "Syntactic Definitions and Abbreviations").

(The symbol `:=` means "is defined as").

aexpr := an arithmetic expression

avar := an arithmetic variable (real or integer)

sexpr := a string expression

numaryname := the name of a numeric array

LabSoft commands will always be presented in this manual in the following way:

X.X **COMMAND WORD**

COMMAND WORD = required argument [,optional arguments]

-or-

COMMAND WORD ,(required parameter) = expression

The command word and any required arguments or parameters will appear immediately under the bold type listing of the command word itself. Optional arguments will appear enclosed in brackets beside the required argument. Optional parameters will be listed as a group just after the definition of the command.

As is true in all ISAAC documentation, we will call your attention to information of more than passing interest by using the three types of call-outs shown below:

NOTE Indicates that this information is of particular interest, even though it may not otherwise have been obvious.

CAUTION Indicates that this is some action or sequence
********* of actions which may have unforeseen consequences and which should be attempted with caution.

WARNING INDICATES SOMETHING WHICH POSES A SERIOUS
!!!!!!! DANGER TO EITHER THE PROGRAM OR THE HARDWARE.
THIS IS SOMETHING TO BE AVOIDED AT ALL TIMES!

In general, a **Caution** will alert you to something which is likely to "hang" or "crash" a program. When a program **hangs**, it is usually possible to recover and return to command level with the program intact by using the RESET key. Recovering from a **crash**, on the other hand, will usually require a complete re-booting of Lab-Soft, with the consequent loss of RAM contents.

A **WARNING**, if ignored, will have far more serious consequences (i.e., damaged hardware, erased disks, etc.).

Every attempt has been made to ensure that this manual is concise, thorough, and easy to read. This manual (like all good documentation) will be periodically revised and updated to correct errors of commission and omission. We welcome your comments and suggestions concerning this manual and the rest of the ISAAC Reference Library.

&&&



CHAPTER 2

THE LABSOFT MASTER DISK

- 2.1 Booting The Disk
- 2.2 Where LabSoft Will Live
- 2.3 Booting LabSoft From Your Own Programs
- 2.4 LabSoft's Relationship to Applesoft
- 2.5 The LabSoft Utilities Disk

CHAPTER 2: THE LABSOFT MASTER DISK

LabSoft is supplied as an object file (in two versions) on the LabSoft Master Disk. This disk also contains the SETCLOCK program (for setting the real-time clock) and an ASCII character set for use with LabSoft's & LABEL command. The HELLO program on the LabSoft Master Disk is designed to load and initialize the proper LabSoft object file for your system.

ISAAC and LabSoft will work with the following varieties of Apples.

- A. Apple II (Integer BASIC) with Applesoft ROMcard in Slot #0.
- B. Apple II (Integer BASIC) with Language Card in Slot #0.
- C. Any Apple II Plus (Applesoft BASIC).

2.1 Booting The Disk

Boot the LabSoft Master Disk just the way you would boot any other disk.

If your Apple has an autostart ROM, turn off system power and insert the LabSoft Master Disk in the "boot" drive (usually Drive #1, Slot #6). Turn on system power and the disk will boot. You may also boot the disk from command level (with the Apple on and either a] or > prompt visible) by placing it in the boot drive and typing:

PR#6

If your Apple has an "old" monitor ROM, insert the LabSoft Master Disk in the "boot" drive, turn on system power, and (when the * prompt appears) type:

6 CTRL-P

Booting LabSoft will:

- A. Load and initialize DOS
- B. Run the LABSOFT HELLO program
- C. Load and initialize LabSoft
- D. Load Applesoft BASIC into the Language Card if one is present in slot #0 (unless your system has Applesoft in ROM).

2.2 Where LabSoft Will Live

LabSoft exists on the disk as two object files (listed in the catalog as LABSOFT.RAM.OBJ and LABSOFT.ROM.OBJ) which will automatically be loaded and initialized in memory at the most advantageous (memory-efficient) location. Depending on the configuration of your Apple, this could be any one of the following places:

- A. In any Apple II with Integer BASIC on the main board and an Applesoft ROMcard or RAMcard (language card) in Slot #0, LABSOFT.ROM.OBJ will live in program memory, just under DOS, (leaving three DOS buffers). Booting the disk will also load Applesoft BASIC into the language card.
- B. In an Apple II Plus (Applesoft BASIC on the main board) with no Language Card in Slot #0, LABSOFT.ROM.OBJ will live in program memory, as above.
- C. An an Apple II Plus with a RAMcard (or Language Card) in Slot #0, LABSOFT.RAM.OBJ will be loaded into the Language Card. This is the preferred configuration, since it leaves 38K of RAM for program memory and HIRES screens, 8K more than will be available with any other configuration.

NOTE Since LabSoft is an extension of Applesoft, it will not run unless Applesoft is installed in either ROM or RAM.

2.3 Booting LabSoft From Your Own Programs

Once a LabSoft object file has been loaded and initialized, the language will reside in memory

until the Apple's power is turned off. LabSoft programs may be written, run, and saved to disk just like Applesoft programs. In many cases, you'll want to write programs and save them on disks that will also boot LabSoft automatically when they are run. This will save the time that would otherwise be required to boot up the Master Disk, then run a program saved to another disk. The LABSOFT HELLO program (since it's heavily REMarked) contains all the instructions you'll need to "lift" the appropriate sections of LABSOFT HELLO and incorporate them into your own HELLO, one which you can use to INITialize the disks on which you plan to save your LabSoft programs.

NOTE

Any of the files on the Master Disk can be copied onto any other disk using the FID program on the Apple DOS 3.3 System Master. Instructions for using FID can be found in the Apple DOS Manual.

NOTE

Cyborg provides one backup copy of the LabSoft Master Disk and one backup copy of the LabSoft Utilities Disk. The LabSoft Master Disk can be copied using any Apple disk copying program. We recommend that an additional working copy of the Master Disk be made as soon as possible after receiving the system. An original should then be stored in a safe place and re-copied when necessary.

2.4

LabSoft's Relationship to Applesoft

LabSoft is an extension of Applesoft BASIC. The **&** (ampersand) which precedes all LabSoft commands is Applesoft's expansion character. When Applesoft encounters an ampersand, it executes an unconditional jump to memory location \$3F5. What it finds there (assuming LabSoft has been properly loaded and initialized) is another jump vector, directing it to the memory location where the LabSoft object file resides.

With LabSoft loaded and initialized, Applesoft programs can be written and run in the normal way. Any Applesoft command may be included in any LabSoft program. If LabSoft has not been loaded and initialized, the Apple will "beep"

and return a ?SYNTAX ERROR message whenever it encounters a LabSoft command.

2.5

The LabSoft Utilities Disk

In addition to the LabSoft Master Disk, you will also be supplied with a disk containing a number of useful utility and demonstration programs. These are all self-documenting and (since they may be changed from time to time without notice) will not be discussed further here. A print-out of the instructions for these utilities can be found in Appendix U of this manual.

Now that you know something about how LabSoft works, we can safely continue with a full description of what it is.

&&&



CHAPTER 3

THE UTILITY COMMANDS

- 3.1 The Beeper/Buzzer Commands
- 3.2 & PAUSE
- 3.3 & ARRAYCLR
- 3.4 & OFFERR
- 3.5 & ERRPTCH
- 3.6 & SLOT #
- 3.7 & FMTDFLT

CHAPTER 3: THE UTILITY COMMANDS

These twelve LabSoft commands are grouped under this heading because they are liable to occur in almost any LabSoft program. Some of these commands simply execute frequently used Apple II system calls in a more convenient way. Others support or enhance the functions of other LabSoft instructions.

3.1 The Beeper/Buzzer commands

& BEEP
& BEEP ON
& BEEP STOP

These commands control the "beep" tone of the ISAAC audio (Beeper/Buzzer) subsystem. & BEEP toggles the beep tone for .1 second. & BEEP ON toggles the beep tone on and keeps it on until an & BEEP STOP command is encountered, or the (RESET) key is pressed. & BEEP STOP will stop an ISAAC beep tone if one is being generated; otherwise the command will be ignored.

NOTE

The ISAAC audio subsystem, unlike that of the Apple, is a dedicated hardware item and does not rely on CPU timing loops to generate its tones. Program execution need not pause when a Beeper/Buzzer command is executed. Beep and buzz tones may be used separately or together.

& BUZZ
& BUZZ ON
& BUZZ STOP

These commands control the "buzz" tone of the ISAAC audio subsystem. They operate precisely like their & BEEP counterparts.

3.2 & PAUSE

& PAUSE = aexpr

& PAUSE pauses program execution for the number of seconds specified by aexpr (aexpr must be in the range 0-6553.5, or an ?ILLEGAL QUANTITY ERROR will result). The resolution of aexpr cannot be greater than .1 second; extra digits will be ignored. For example, a statement of the form & PAUSE = 3.5 will pause program execution for 3.5 seconds. A statement of the form & PAUSE = 3.5292 will also pause program execution for 3.5 seconds; digits to the right of the tenths place will be ignored.

NOTE An & PAUSE instruction may be interrupted by a CTRL-C from the Apple keyboard.

NOTE & PAUSE derives its timing from the Apple's crystal clock, which, although extremely accurate, operates at a frequency which does not divide down evenly into seconds. A "LabSoft Second", as measured by & PAUSE, will typically be in the vicinity of 1.0034 seconds. If long & PAUSE's are programmed, you may notice that the delays are somewhat different than specified. & PAUSE is still extremely accurate, and although a programmed delay of 100 seconds may actually take 100.3 seconds, it will take 100.3 seconds every time it is run.

3.3 & ARRAYCLR

& ARRAYCLR, numaryname

& ARRAYCLR clears to zero all elements of the numeric array specified by numaryname. Use only the name of the array; do not include any of its subscripts. In the example below, line 10 DIMensions array A; line 200 clears it (leaving it DIMensioned as shown in line 10) using & ARRAYCLR.

```
10 DIM A(99)
```

```
////////////////////
```

```
200 & ARRAYCLR, A
```

NOTE

If numaryname is the name of a string array, a ?TYPE MISMATCH ERROR will result. If an array named numaryname has not been previously DIMensioned, an ?OUT OF DATA ERROR will result.

3.4**& OFFERR**

& OFFERR turns off Applesoft's ONERR GOTO status (performs the same function as POKE 216,0). For more information on ONERR GOTO, see your Applesoft Reference Manual.

3.5**& ERRPTCH**

& ERRPTCH executes Applesoft's ONERR GOTO error-handler patch as described in the Applesoft Reference Manual. Because of a minor bug in Applesoft, there is a tendency for the Apple to lose one RETURN address or one NEXT address whenever an error occurs and ONERR GOTO is active. By executing & ERRPTCH as the first line of your error-handling routine (the routine that your ONERR GOTO "goes to"), this problem can be eliminated.

3.6**& SLOT#**

& SLOT# = aexpr

& SLOT# specifies the Apple peripheral slot containing the ISAAC/Apple Interface Card which LabSoft is currently "speaking to". When you load and initialize LabSoft, it expects to find the ISAAC/Apple Interface Card in Apple Peripheral Slot #3. If you can't put the Interface Card in Slot #3 (because you have some other slot-specific card there), you must let LabSoft know which slot the Interface Card is in, using a statement of the form:

& SLOT# = aexpr

where aexpr is the number (0-7) of the Apple Peripheral Slot where the ISAAC/Apple Interface card is installed.

Caution **& SLOT#**, if you need to use it, should come very
***** early in your program, so LabSoft knows where to
find the ISAAC/Apple Interface card. If the card
is not located in Slot #3 and no other location
has been specified, the first occurrence of a
LabSoft command will cause the system to hang or
crash. If you have more than one ISAAC connec-
ted to your Apple, you will have to use **& SLOT#**
to specify the ISAAC that LabSoft is supposed to
access.

3.7 **& FMTDFLT**

& FMTDFLT

& FMTDFLT sets the arguments of all of LabSoft's
format commands to the following default values.

& CNTFMT = 7

& PLTFMT = 1

& BINFMT = 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

& ANAFMT = 0

NOTE

Loading and initializing LabSoft automatically
executes **& FMTDFLT**. Relying on this can some-
times be risky, however, since any format that
may have been specified after the initial "boot"
will take precedence over **& FMTDFLT**. In gene-
ral, it's a good practice to explicitly specify
the channel(s) and/or color(s) you wish LabSoft
to make use of.

Now that we've explained how to make noise and
take care of the inevitable "housekeeping", we
will proceed with a demonstration of how LabSoft
draws graphs.

&&&



CHAPTER 4

THE GRAPHING COMMANDS

- 4.1 & HIRES1
- 4.2 & HIRES2
- 4.3 & FULLSCREEN
- 4.4 & MIXSCREEN
- 4.5 & SCROLLSET
- 4.6 & ALTSET
- 4.7 & OUTLINE
- 4.8 & RETRCE
- 4.9 & PLTFMT
- 4.10 & NXTPLT
- 4.11 & NXTBIN
- 4.12 & BINFMT
- 4.13 & LABEL

CHAPTER 4: THE GRAPHING COMMANDS

The efficient display of data is as important as the efficient acquisition of data. That's why LabSoft's instruction set includes numerous graphing commands designed specifically to facilitate on-line or off-line display of ISAAC I/O data. These commands take advantage of the all of Apple II's high-resolution color graphics capabilities. They have been designed to maximize graphing power and flexibility while minimizing the need to write long (and slow-executing) graphing subroutines in Applesoft.

These are specialized "graphing" commands. They can be used only to format graphs for data display. They are not general-purpose "graphics" commands. Applesoft contains numerous graphics commands, used for plotting points and drawing lines on the high resolution screen. These, like all Applesoft commands, will remain available to LabSoft users and can be integrated with any LabSoft program.

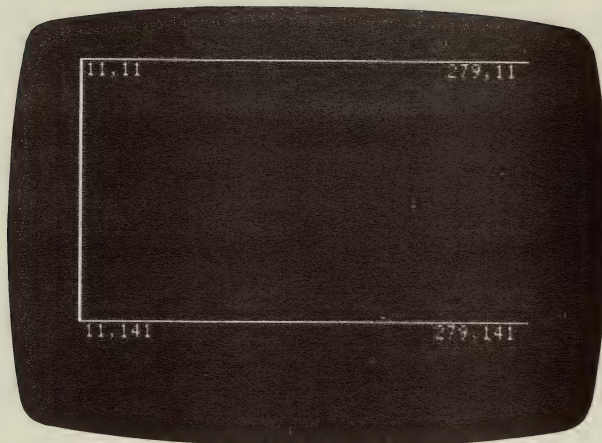
Before we talk about the commands themselves, we'd like to talk a little bit about graphing in general, and how it's done using LabSoft. LabSoft supports two graph types. Each has its own advantages and disadvantages. Learning a little about them will help you to make the best choice for your application.

The Scrolling Graph

The scrolling graph is plotted on the screen in a "window" 128 points high and 265 points across. The top, bottom, and left margins may be displayed as an outline if desired. (Fig. 4:1, next page)

NOTE

The numbers shown on the screen in figures 4:1 and 4:2 represent the co-ordinates of the corners of the graph window. These numbers are not normally displayed.



-FIGURE 4:1-

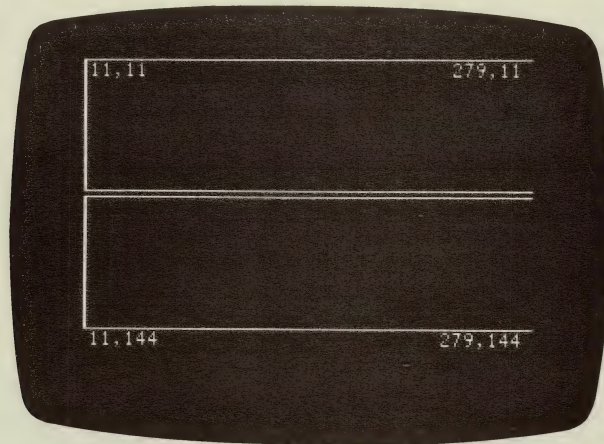
This graph begins scrolling to the left (like the paper in a strip-chart recorder) if more than 265 horizontal points are plotted. The principal advantage of this graph lies in its superior vertical resolution; 128 points. Its disadvantages are that only 265 points across the screen are visible at any one time, and that when the display begins to scroll (which only happens after more than 265 points are plotted), it takes somewhat more time to display each new point.

Think of the scrolling graph as a set of 265 vertical columns (called "pixel" columns), each 128 points high. Each column represents a plotting position, in which one or more data values can be plotted. Normally, the greater the value of the point being plotted, the higher it appears in the column.

When scrolling begins, LabSoft erases the two leftmost pixel columns, shifts all the other points two pixel columns to the left, and plots the 266th point in one of the two rightmost pixel columns. Exactly which of the two columns is used depends on the color of the point. This process continues as long as the graph is forced to scroll.

The Alternating Graph

LabSoft can also display data on a graph which alternates between two "windows". Each window is 64 points high and 265 points across. (Fig. 4:2)



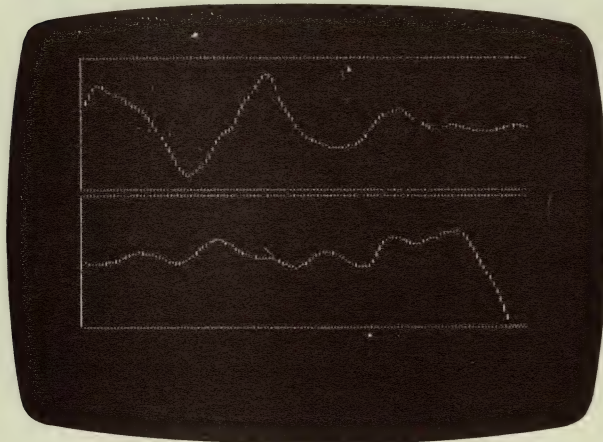
-FIGURE 4:2-

On the alternating graph, the bottom window is actually just an extension of the top window. The first 265 points plotted on this type of graph are plotted in the upper window. The next 265 points are plotted in the lower window. If more than 530 points are plotted, the upper window is erased and plotting is continued there. As more points are plotted, the plot continues in this fashion, alternating between the upper and lower graph windows.

The advantages of this type of graph are that 530 points can be viewed at one time on the screen, and that plotting can be done much faster, since the graph does not scroll. Its disadvantages are that its vertical resolution is compressed (rendering changes in amplitude less apparent), and that very rapid alternation between the graph windows (as when rapidly-acquired data is being displayed in real-time) tends to produce a "strobe" effect that, we'll have to admit, can be pretty tough on the eyes.

Graphing Analog Values

In addition to supporting two different graph types, LabSoft can also graph two different types of values: analog and binary. Analog values vary continuously over a fixed range. When graphed, analog values typically produce curves which rise and fall as the values change over time. LabSoft interprets analog data as one of 4096 possible numeric values and can plot an analog value as one of 128 points (64 if an Alternating graph is specified) in a plotting position (pixel column). LabSoft allows graphing of from one to sixteen analog curves, in any of six colors, on either a scrolling or alternating graph. (Fig. 4:3)



-FIGURE 4:3-

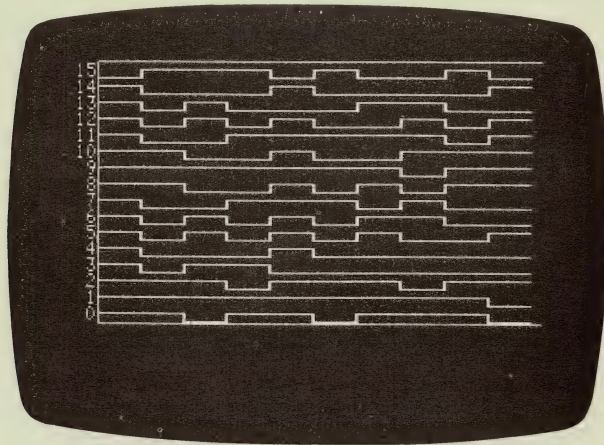
This particular graph displays the output of a piezoelectric device. Because of the large number of data points to be plotted, an alternating graph was chosen.

Graphing Binary Values

In some ways, binary values are a good deal less complex than analog ones. A binary bit can only be high or low. There is no in between. LabSoft can plot values of up to sixteen binary bits (the decimal equivalent of 0 to 65535) on either a scrolling or an alternating graph. There are sixteen possible bit positions on a binary graph. Each bit position occupies roughly one-sixteenth of a pixel column, just enough room to display a bit's value in either of its two possible states. LabSoft allows you to plot either raw binary values or converted analog values in this fashion. (Fig.4:4)

NOTE

Flexible as it is, LabSoft is not flexible enough to allow both analog and binary plotting on the same graph.



-FIGURE 4:4-

This particular graph shows a 16-bit binary value. Bits may be displayed in their "natural" order, or in any other order you wish. The plot representing each bit will be plotted high or low, reflecting the value of that bit at the time of plotting. Bit positions are numbered as shown, although these numbers aren't normally displayed as part of a LabSoft binary graph.

In addition to these three graph types, LabSoft has several formatting and plotting options which take advantage of the Apple II's color graphics capabilities, allowing you to create impressively detailed, easy-to-read graphs. The process is remarkably simple, involving only a few commands, and will be covered in detail after the commands have been properly introduced.

A Note on Memory Management

Any use of the Apple II's high-resolution graphics capabilities requires the setting aside of a certain amount of memory. Each of the Apple's high-resolution screens occupies 8K of memory, so there will not be as much memory available for programs or data when high-resolution graphics are used. There are a number of memory-management schemes which can help you to gain access to all the memory space available in your Apple. We have listed a few of the more useful of these in appendix G of this manual.

Graphing Commands and Graphing Routines

LabSoft's graphing commands are usually used as elements of a graphing routine. These routines can take a number of forms, although the order in which various commands appear within routines is pretty much determined by the functions which the commands perform.

The graphing commands are presented here in the order in which they are most likely to occur in a graphing routine. Of course, you won't ever need to use them all in any graphing routine, but it's a safe assumption that the commands in any well-written graphing routine will follow the order in which they are presented here.

After we have introduced you to the graphing commands and discussed them individually, we will demonstrate the ways in which they may be assembled into different types of graphing routines.

4.1 **& HIRES1**

& HIRES1

& HIRES1 displays the Apple II high resolution screen #1 **without erasing** its contents. The normal four line "text window" is also displayed. Similar to the Applesoft HGR command, but does not clear the screen to black. This command is useful for entering the HIRES graphics mode without erasing the current contents of screen #1.

4.2 **& HIRES2**

& HIRES2

& HIRES2 displays the Apple II high resolution screen #2 (without erasing its contents), allowing full screen graphics only (no text window). **& HIRES2** is similar to the Applesoft HGR2 command, but does not clear the screen to black. Use this command to return to the second HIRES screen without erasing its current contents.

NOTE As a general rule, the first program line of any graphing routine should contain one of these commands, or their Applesoft counterparts, HGR and HGR2.

4.3 **& FULLSCREEN**

& FULLSCREEN

& FULLSCREEN switches any graphics display (LORES, HIRES #1, HIRES #2) to full screen graphics mode with no text window. Executes POKE -16302,0.

NOTE If an **& FULLSCREEN** command is executed when display is in the text mode, or when full screen graphics are already being displayed, it will have no effect.

4.4 **& MIXSCREEN**

& MIXSCREEN

& MIXSCREEN switches the current **HIRES** screen from full screen graphics to a mixed screen of graphics and four-line text window. Executes **POKE -16301,0**.

NOTE **& MIXSCREEN** should not be used if high-resolution graphics screen #2 is being displayed (**& HIRES2** or **HGR2**), since the text lines will be taken from text page #2. Text page #2 is not readily available on the Apple II, and an attempt to display it will probably produce four lines of incomprehensible program material which will not respond to input from the keyboard.

NOTE If an **& MIXSCREEN** command is executed when display is in the text mode, or when mixed-screen graphics are already being displayed, it will have no effect.

4.5 **& SCROLLSET**

& SCROLLSET

& SCROLLSET initializes a scrolling graph display for either analog or binary graphing (not both). It clears the scrolling graph window to black, and resets the horizontal position pointer to the first plotting position (the leftmost pixel column of the scrolling graph window). Only that part of the high-resolution screen occupied by the scrolling graph window will be erased by this command.

NOTE If you use this command while no high-resolution screen is being displayed, the command will be executed, but its results will not be visible unless (or until) a high-resolution graphics screen has been selected using any of the following commands: **HGR**, **HGR2**, **& HIRES1**, **& HIRES2**.

4.6

& ALTSET

& ALTSET

& ALTSET initializes an alternating graph display for either analog or binary graphing (not both). It clears the upper and lower graph windows to black and resets the horizontal position pointer to the first plotting position (the leftmost pixel column of the upper graph window). Only that part of the high-resolution screen occupied by the graph windows will be erased by this command.

NOTE

If you use this command while no high-resolution screen is being displayed, the command will be executed but its results will not be visible unless (or until) a high-resolution graphics screen has been selected using any of the following commands: **HGR**, **HGR2**, **& HIRES1**, **& HIRES2**.

NOTE

Although the alternating graph has only half the vertical resolution of the scrolling graph, it offers twice the horizontal resolution. In addition, the display can be updated much faster than the scrolling graph, making this the preferred format for fast, real-time displays.

4.7

& OUTLINE

& OUTLINE

& OUTLINE outlines the currently defined graph type in the currently defined **HCOLOR**. Be sure to specify a graph type **before** using this command. If this command is executed before a graph type has been defined (using either **& ALTSET** or **& SCROLLSET**), LabSoft will not know which type of outline to draw and will either draw no outline at all, or will draw some portion of one or both outlines.

NOTE

If the currently specified HCOLOR is black (0), no outline will be drawn. HCOLOR may be set to any of the following values:

HCOLOR=	COLOR	HCOLOR=	COLOR
0	black	4	black
1	green	5	orange
2	blue	6	purple
3	white	7	white

(Older Apples may only allow use of colors 0-3)

NOTE

To outline a graph, this command performs the equivalent of the following Applesoft instructions.

For a scrolling graph:

```
]HPLOT 279,11 TO 11,11 TO 11,141 TO 279,
141 : HPLOT 12, 12 TO 12, 140
```

For an alternating graph:

```
]HPLOT 279,11 TO 11,11 TO 11,77 TO 279,
77 : HPLOT 12,12 TO 12,76 : HPLOT 279,
79 TO 11,29 TO 11,144 TO 279,144 :
HPLOT 12,80 TO 12,143
```

4.8**& RETRCE****& RETRCE**

& RETRCE resets the horizontal position pointer of the currently defined graph type (& SCROLLSET or & ALTSET) to the first plotting position without clearing the screen. Using this command, one graph may be plotted and subsequent graphs plotted (retraced) over it without erasure.

4.9**& PLTFMT**

& PLTFMT = aexpr1 [, aexpr2..., aexprN]

Aexpr must be an integer in the range of 0-7, or an ?ILLEGAL QUANTITY ERROR will result. Arguments must be separated by commas. & PLTFMT specifies the Applesoft HCOLOR(s) which will be used for plotting analog values. It has no

effect on the plotting of binary values. & PLTFMT provides the **only way** to define the color of an analog graph plot. This command sets the analog plotting format pointer to the first value specified in the format. If no color is specified, a ?SYNTAX ERROR will result.

If only one analog value is to be plotted at a given plotting position, then aexpr1 will determine the color of that plot. If more than one analog value is to be plotted at each plotting position, LabSoft will plot the first value in the color specified by aexpr1, then plot the next value in the color specified by aexpr2, and so on through aexprN. The order in which these analog values are to be plotted must be specified using & NXTPLT commands.

NOTE

Graphing routines using & PLTFMT will plot as many points at a given horizontal position as there are colors in the most recently defined & PLTFMT. If you attempt to specify more than 16 values in the argument of & PLTFMT, a ?SYNTAX ERROR will result.

NOTE

Only one plotting format can be in effect at a given time. The only way to change a plotting format is with an & PLTFMT command. Typing NEW won't do it. So it's a good idea to explicitly specify a new & PLTFMT whenever there is any doubt as to whether a previously specified one is the one you want.

The following Apple II HCOLOR numbers may be used in the argument of & PLTFMT.

HCOLOR=	COLOR	HCOLOR=	COLOR
0	black	4	black
1	green	5	orange
2	blue	6	purple
3	white	7	white

(Older Apples may only allow use of colors 0-3)

NOTE

HCOLORs 0 and 4 (black) may be specified but will not be plotted. If, in a series of plotted analog values, there is one that you don't wish displayed, "plotting" it in black is a way to accomplish this.

4.10 **& NXTPLT**

& NXTPLT = aexpr

Aexpr must be in the range of 0-127 or an ?ILLEGAL QUANTITY ERROR will result. The argument of & NXTPLT specifies the "next point" of **analog** data to be plotted on the currently defined graph type (scrolling or alternating). This point will be plotted in the color specified by the most recently defined analog plot format (& PLTFMT). There should be a one-to-one correspondence between the number of & NXTPLT commands in a graphing routine and the number of arguments of the most recently specified & PLTFMT. & NXTPLT has no effect on binary graphing.

NOTE LabSoft has an I/O parameter, (GA), which performs a function similar to & NXTPLT. (GA) will be discussed, along with the other I/O parameters, in the next chapter of this manual.

Caution If & NXTPLT appears in a program before the
***** first occurrence of one of the high-resolution graphics selection commands (HGR, HGR2, & HIRES1, & HIRES2), or before one of LabSoft's graph types (& SCROLLSET, & ALTSET) has been selected, the system may hang or crash.

4.11 **& NXTBIN**

& NXTBIN = aexpr

Aexpr must be in the range of 0-65535 or an ?ILLEGAL QUANTITY ERROR will result. The argument of & NXTBIN specifies the "next binary value" to be plotted on the currently defined graph type (scrolling or alternating). Plotting is done in the most recently defined HCOLOR. The order in which the bits are displayed is set by the most recently executed & BINEMT command.

Caution If & NXTBIN appears in a program before the
***** first occurrence of one of the high-resolution graphics selection commands (HGR, HGR2, & HIRES1, & HIRES2), or before one of LabSoft's graph types (& SCROLLSET, & ALTSET) has been selected, the system may hang or crash.

4.12 **& BINFMT**

& BINFMT = aexpr1 [,aexpr2,... ,aexprN]

Aexpr must be an integer in the range of 0-15 or an ?ILLEGAL QUANTITY ERROR will result. Argument expressions must be separated by commas. **& BINFMT** sets the bit numbers, and (implicitly) sets the number of bits to be plotted by the **& NXTBIN** command. If more than 16 arguments appear in this command, a ?SYNTAX ERROR will occur.

Bit positions on LabSoft's binary graph start at the bottom (aexpr1) and go to the top (aexpr16). A statement of the form

& BINFMT = 10,11,12

would tell LabSoft to plot bits 10, 11, and 12 of a binary value at bit positions 1, 2, and 3. Remember that LabSoft numbers binary bits 0-15, from right (least significant bit) to left (most significant bit).

4.13 **& LABEL**

& LABEL = sexpr AT aexpr1, aexpr2

& LABEL allows you to label graphs or otherwise put text on the Apple II's high-resolution graphics screen. **& LABEL** positions the center of the first character of the specified string at the coordinates specified by aexpr1 and aexpr2. The following conditions apply to the use of this command:

- A. Sexpr must be a valid Applesoft string expression. It must contain less than 40 characters or a ?STRING TOO LONG ERROR will result. If sexpr contains any control characters (or characters whose ASCII values are less than 33 or greater than 127), these characters will not be printed.
- B. The coordinates of the center of the first character of the string should be specified by aexpr1 (the horizontal coordinate) and aexpr2 (the vertical coordinate).

- C. If any of the following relationships exist,
an ?ILLEGAL QUANTITY ERROR will result.

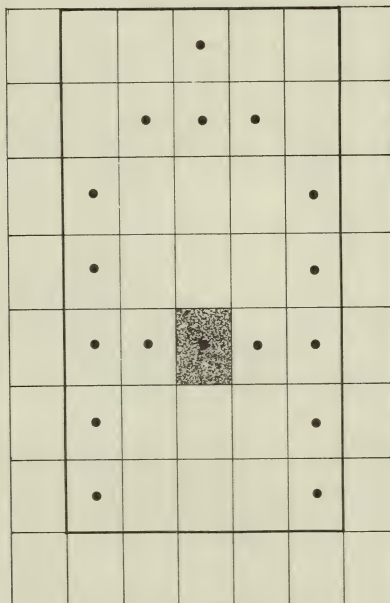
aexpr1 < 4 OR
aexpr1 > 276 OR
aexpr2 < 4 OR
aexpr2 > 187

- D. If the center of any of the characters in
the string to be printed has coordinates
that meet any of the above conditions, a
?STRING TOO LONG ERROR will result.

NOTE

It is possible to load and initialize LabSoft
without loading the ASCII character set used by
& LABEL. Each time this command is executed, it
checks to make sure that a valid character set
is currently in memory. If none exists, this
command will be ignored. The LabSoft HELLO
program contains full instructions for loading
this character set.

When we refer to the coordinates of "the center"
of a character, we are talking about a very
specific point. ASCII characters drawn on the
high resolution screen are assembled as a dot
matrix which is eight dots high and seven dots
wide. (Fig. 4:5)



-FIGURE 4:5-

The "A" in Fig. 4:5 is drawn, like most ASCII characters, on a seven dot high by five dot wide matrix. The bottom row is used only those ASCII characters with descenders, (g, y, p, q, and the comma). For that reason, the "center" of a character is considered to be the center of the 5x7 matrix most often used. In the above illustration, that "center" dot has been shaded.

NOTE

Labels printed in a scrolling graph window will scroll along with the rest of the graph, eventually scrolling off the screen. Labels printed in an alternating graph window will be erased with each cycle of the graph.

Using & LABEL, you can print label strings anywhere on either of the Apple II's high resolution screens. Any printable ASCII character or character string may be specified. There are several ways to specify the desired string.

- Include the string, in quotes, in the argument of & LABEL.

```
100 & LABEL = "ISAAC" AT 5, 140
```

- Specify the string in an earlier program line, give it any legal variable name, and include that variable name in the argument of & LABEL.

```
90 X$ = "ISAAC"  
100 & LABEL = X$ AT 5, 140
```

- Or, if you need to specify a character that doesn't exist on the Apple keyboard (like a lower-case character, for instance), you may use the ASCII character code found, among other places, in your Applesoft Programming Manual. For example

```
100 & LABEL = CHR$(97) AT 5,140
```

prints a lower-case "a" in the upper middle of the screen.

NOTE

The numbers shown on figures 4:1, 4:2, and 4:4 of this manual were put there using the & LABEL command.

You may also be interested in generating your own character sets for use with this command. Appendix I of this manual has more on this. Appendix M contains a character-conversion subroutine applicable to this command.

NOTE

If you attempt to & LABEL at vertical coordinates greater than 156 while you are in the mixed graphics and text display mode (& MIXSCREEN), the labels will be obscured by the text window. To label in these positions, you must be in the full screen graphics mode (& FULLSCREEN).

Analog Graphing Routines

Now that the graphing commands have been properly introduced, we'd like to take a little time to show you how they are organized into graphing routines.

The first part of any graphing routine should specify a high-resolution screen to work on, a graph type, and more than likely, an outline. The following program lines set up an alternating graph outlined in white.

```
1000 HGR
1010 & ALTSET
1020 HCOLOR = 3
1030 & OUTLINE
```

In the interest of brevity, assume that there are several 500 point data arrays, named DA, DB, and DC already in memory, and that each array contains no value greater than 127. (Any value greater than 127 must be scaled to fall within plottable range). The easiest way to plot one of these arrays (in green) would be to specify an & PLTFMT, then construct a three line loop using & NXTPLT.

```
1040 & PLTFMT = 1
1050 FOR N = 0 TO 499
1060 & NXTPLT = DA(N)
1070 NEXT N
```

Each time the loop is executed, a point is taken from DA and plotted at the next horizontal loca-

tion in the color specified by the currently active & PLTFMT. Normally, LabSoft draws lines connecting the points of a given plot. Appendix E of this manual shows you how to change this feature, if you need to.

Additional arrays could be plotted using several different methods. & RETRCE could be used and the program continued as shown below.

```
1080 & RETRCE
1090 FOR N = 0 TO 499
2000 & NXTPLT = DB(N)
2010 NEXT N
```

These lines would plot array DB in green (the currently active plotting format) unless a new color was specified, using & PLTFMT, somewhere between lines 1070 and 2000. Another way to accomplish the same thing (plotting arrays DA and DB in green) would be to use & NXTPLT and & PLTFMT and rewrite the program as shown below.

```
1040 & PLTFMT = 1,1
1050 FOR N = 0 TO 499
1060 & NXTPLT = DA(N)
1070 & NXTPLT = DB(N)
1080 NEXT N
```

The first time a & NXTPLT command is encountered in a loop like the one above, LabSoft moves to the next horizontal plotting position, checks the plotting format, and plots a point specified by the argument of that & NXTPLT in the color specified by the first value in the plotting format. For each subsequent occurrence of & NXTPLT within the plotting loop, LabSoft plots the value specified by the argument of that & NXTPLT in the the color specified by the next value in the plotting format.

& PLTFMT and & NXTPLT can be combined in this way to plot different arrays (e.g., DA, DB, and DC) in different colors. The addition of one more number and one more line to this program would do the trick.

```
1040 & PLTFMT = 1,2,5
1050 FOR N = 0 TO 499
1060 & NXTPLT = DA(N)
```

```
1070 & NXTPLT = DB(N)
1090 & NXTPLT = DC(N)
2000 NEXT N
```

NOTE

If the & RETRCE method is used to display multiple plots on one graph, a new & PLTFMT command (with only one argument) will have to be executed before each & RETRCE command if you need to color-code your plots.

NOTE

Because of the way the Apple generates video color information, it is not possible to plot every color in every pixel (plotting position). A given color (other than white) will plot only in **every other** pixel. In addition, when many graphs are plotted simultaneously, the crossing of two plots may occasionally cause one plot to change color. This is partially because of the way in which the Apple II derives its HIRES colors, and partially because of the algorithm LabSoft uses to determine where to plot. If the algorithm was changed to trap all incidence of this "complementary color carry-over", graph scrolling would be extremely slow. You shouldn't have to worry too much about this phenomenon, since it rarely occurs and, when it does, lasts only as long as both plots maintain their vertical positions. For the same reasons, scrolling graphs that actually scroll (i.e., those that contain more than 265 points) should not be plotted in either of the **white** colors (3,7). When the graph scrolls, the white plots will "carry over" in their color complement.

Binary Graphing Routines

Binary graphing routines are in many ways similar to their analog counterparts. In most cases, they will start in exactly the same way.

```
1000 HGR
1010 & SCROLLSET
1020 HCOLOR = 3
1030 & OUTLINE
```

Due to the more limited complexity of binary numbers, these routines tend to follow a more fixed pattern. It is possible to plot as many bits of a value (up to 16) as you wish, and to

display the bits in any order you wish. Binary plotting is always done in the currently specified HCOLOR. The following loop plots the values of data array DA in binary form, with the least significant bit displayed at the bottom of the graph window and the most significant bit displayed at the top.

```
1040 & BINFMT = 0,1,2,3,4,5,6,7,8,9,10,11,12,13
      ,14,15
1050 FOR N = 0 TO 199
1060 & NXTBIN = DA(N)
1070 NEXT N
```

If you only wanted to display the first four bits of this value, and wanted to plot them in a different color than the graph outline, a few small changes would be necessary.

```
1040 & BINFMT = 0,1,2,3
1050 HCOLOR = 1
1060 FOR N = 0 TO 199
1070 & NXTBIN = DA(N)
1080 NEXT N
```

NOTE Bits may be displayed in any order. If line 1040 above had been written as

```
1040 & BINFMT = 3,2,1,0
```

the order (bottom to top) in which the bits were plotted would be reversed. Displaying binary bits in their "natural" order may be inappropriate for certain applications, depending upon the information represented by the bits. Because it can be difficult, for example, to visually compare bits across many state lines, you may want to specify that the value of a given bit be plotted in every fourth bit position. This type of scheme can make visual bit-state comparisons easier.

NOTE Since & NXTBIN plots each new binary value in the next horizontal plotting position, bits which change state very rapidly (low-order bits, usually) will be displayed as a solid bar across their bit position.

This Page Intentionally Left Blank



CHAPTER 5

THE INPUT/OUTPUT PARAMETERS

- 5.1 (TV)
- 5.2 (DV)
- 5.3 (AV)
- 5.4 (TH)
- 5.5 (D#)
- 5.6 (C#)
- 5.7 (FU)
- 5.8 (XM)
- 5.9 (AM)
- 5.10 (GA)
- 5.11 (GB)
- 5.12 (PR)
- 5.13 (RT)
- 5.14 (SW)
- 5.15 (CV)
- 5.16 (W#)

CHAPTER 5: THE INPUT/OUTPUT PARAMETERS

LabSoft's Input/Output (I/O) commands and parameters are the real heart of the LabSoft instruction set. These commands and parameters accomplish the primary purpose of the ISAAC system: acquiring data (Input) from the real world and sending messages (Output) back, all via ISAAC's I/O hardware. The I/O parameters covered in this chapter are an integral part of LabSoft's command structure.

Structure of the I/O Commands and Parameters

I/O commands have a structure which differs from the more familiar BASIC and Applesoft commands, and even from other LabSoft commands. For every LabSoft I/O command, there is at least one required parameter, as well as a number of parameters which are optional. The structure of the LabSoft commands is somewhat like that of Apple DOS commands. As you know, most DOS commands can be used with various parameters, such as Drive#, Slot#, and address. LabSoft's I/O parameters, however, come in much greater variety. They provide an easy way to tailor any LabSoft command to the needs of your particular application.

There are two parts to every I/O command: the command word, and the parameter specification. Schematically, an I/O command looks like this.

& command word, (parameter) ,[parameters]

That is to say, the command word is followed by a comma, at least one required parameter, and whatever optional parameters are needed, all separated by commas. A numeric value or variable name must be assigned to all required parameters and most optional parameters. For example, all simple input commands (commands which get an input value from ISAAC) require that you specify the name of the target variable (i.e., the variable to which you want the value

returned). Optional parameters allow you to modify or change the values returned or written by a command.

NOTE

There are two LabSoft commands that have more than one required parameter. & RDEV and & WRDEV are expansion commands, and since they are designed to access a wide variety of potential hardware, they are not in themselves very specific. That's why they require extra parameters. These commands are covered in Appendix X of this manual.

All required and most optional parameters use parameter labels, which take the form of simple assignment statements. The format for these statements is:

,(parameter label) = expression

The exact nature of expression varies from parameter to parameter. The parameters and their syntactic requirements are listed as a group in the LabSoft Quick Reference Guide, as well as in Appendix K of this manual. In this chapter, we will give you a somewhat more detailed view of the parameters and suggest how to use them to the best advantage.

The LabSoft I/O Parameters

5.1 (TV) Target Variable

(TV) = avar

This is a **required** parameter for all simple (not matrixed) **input** commands. Avar is the Applesoft arithmetic variable in which LabSoft will store the results of the input operation.

5.2 (DV) Data Value

(DV) = aexpr

This is a **required** parameter for all simple (not matrixed) **output** commands. After possible modifications (by any masks and optional functions), this value will be output via ISAAC's hardware.

Aexpr can be any valid arithmetic expression, but it must be scaled by an optional function to fall within the range 0 to 65535 before it can be output. Final output values not within this range will generate an ?ILLEGAL QUANTITY ERROR.

NOTE Although all values in the integer range of 0-65535 are legal output values, an output value with more significant binary bits than are allowed for the designated output device (the 12-bit D/A converters, for instance) will either "wrap around" (be given a value MOD the maximum significance allowed by the device in question) or have the extra significance ignored.

5.3 (AV) Array Variable

(AV) = numaryname

This is a **required** parameter for all **matrix** I/O commands (see the note on Matrix operations at the end of this chapter). When (AV) is used with an **input** command, each value input and processed is stored in the next element of the array named numaryname. When (AV) is used with an **output** command, the array named numaryname is used as the data array for output. Each element is taken in sequence from the array and used as the data value for one output.

NOTE Unless otherwise dimensioned (using AppleSoft's DIM statement), arrays contain 10 elements, numbered 0 to 9.

About Data Arrays

In LabSoft, (as in BASIC) the "first" element of a data array is defined as element 0 of the fastest-ascending (leftmost) dimension. The "next" element used will be element 1 of that dimension, and so on. When there are no more elements in the fastest-ascending dimension, then element 0 of the second fastest-ascending dimension is used.

NOTE One way to think of Array Variables, at least as they relate to LabSoft analog I/O operations, is to consider that, of the two array elements, the

first represents the number of channels you're dealing with and the second represents the number of samples you want to take from (or outputs you want to send to) each channel. An array DIMensioned (3,5) will hold 5 samples from each of 3 channels.

The following example may help clarify the order in which array elements are used (i.e., to store data in or take data from). The elements of an array DIMensioned as follows

```
10 DIM ARRAY (2,3)
```

will be used as shown in the following table, where Subscripted Element ARRAY (0,0) is used first (is Element #1) and Subscripted Element ARRAY (2,3) is used last.

Element #	Subscripted Element	
1	ARRAY	(0,0)
2	ARRAY	(1,0)
3	ARRAY	(2,0)
4	ARRAY	(0,1)
5	ARRAY	(1,1)
6	ARRAY	(2,1)
7	ARRAY	(0,2)
8	ARRAY	(1,2)
9	ARRAY	(2,2)
10	ARRAY	(0,3)
11	ARRAY	(1,3)
12	ARRAY	(2,3)

NOTE

Numaryname can only be the name of a previously dimensioned arithmetic array. If the array has not been dimensioned prior to the execution of an I/O command where it is specified as the array variable, then an ?OUT OF DATA ERROR will occur. A ?SYNTAX ERROR will result if array subscripts are included in the definition. For the example array above, the required (AV) parameter would be specified as

```
(AV) = ARRAY
```

It could become confusing (to you as well as your computer) if your programs use simple real variables with the same names as array variables. It's a good practice to avoid.

5.4 (TH) THreshold value

(TH) = aexpr

This is an optional parameter for all input commands which are being used as triggers in "& LOOK FOR... THEN" expressions (see the note on Triggered Operations at the end of this chapter). Aexpr must be in the range of 0-65535 or an ?ILLEGAL QUANTITY ERROR will result. The default value of this parameter (the one which LabSoft will assume you want if (TH) is omitted from an & LOOK FOR... THEN... expression) is 0. Since one function of trigger commands is to LOOK FOR... a specific input value, if (TH)=0 any input value of the type specified will execute the THEN... portion of the command. The function of the (TH) parameter is to specify a particular value to be LOOKed FOR. Proper use of this parameter is essential if you wish to effectively utilize triggered operations. For more information on the whole business of & LOOK FOR... THEN..., read Chapter 10 of this manual.

NOTE Use of (TH) with any command other than the trigger of & LOOK FOR...THEN... will generate a ?SYNTAX ERROR.

5.5 (D#) Device

(D#) = aexpr

Aexpr must be an integer in the range of 0 - 15 or an ?ILLEGAL QUANTITY ERROR will result. This is an optional parameter for most LabSoft I/O commands. It specifies the device which the command will access. All LabSoft analog and binary I/O commands must include a (D#) parameter in the command line, specifying the number of the I/O device to be accessed. Each ISAAC device slot is numbered. To access a device in a given slot, set aexpr equal to the number of that slot.

NOTE The model 91A ISAAC has analog I/O, binary I/O, and counter devices which LabSoft will access by default when no (D#) is specified in an I/O command line. Consult the device map in the ISAAC 91A System Reference for further details.

5.6 (C#) Channel

(C#) = aexpr

Aexpr must be in the range of 0-255 or an ?ILLEGAL QUANTITY ERROR will occur. This parameter allows you to specify the channel to be used in an analog or counter I/O operation. Note that the number of channels available varies depending on which I/O subsystem is being accessed. Consult your ISAAC System Reference and/or ISAAC I/O Module User Manual for further information on channels and channel access.

NOTE

If the value of aexpr is greater than the highest channel number of the device being accessed, the channel selected will be the one specified MOD the total number of channels incorporated in that device. In effect, what happens is that the argument (aexpr) "wraps around" when it exceeds the highest allowable channel. For example, specifying (C#) = 6 in a command line accessing a four-channel device would result in channel #2 being selected.

If this parameter is not specified, the channel number for an operation will be taken from the most recently specified channel format. If the operation is an analog input or output operation, the channel will be taken from the most recently executed & ANAFMT command. If the operation is a counter input, the channel will be taken from the most recently executed & CNTFMT command

NOTE

The LabSoft command & FMTDEFLT (which resets all LabSoft formats to certain default values) is executed each time the LABSOFT HELLO program is run. However, relying on these default formats is not recommended. The best way to explicitly specify a channel-access format (and reset the format pointer to the first value in that format) is to execute the appropriate command (& ANAFMT, & CNTFMT). Only one format of each type can be active at a given time. The most recently specified format will always be used.

5.7

(FU) FUnction

(FU) = FN name (avar)

Most of LabSoft's I/O commands allow you to specify an optional function. In an input command, a function can be used to modify data after it is input from hardware, and before it is stored in the target (or array) variable. In an output command, a function can be used to modify the data value to be output after it is fetched from the data value expression or array variable, and before it is output to hardware.

(FU) can be any valid Applesoft function. (See the Applesoft Reference Manual for more on functions.) The function specified must have been previously defined or an ?UNDEFINED FUNCTION ERROR will result.

Using (FU) With Input Commands

In input commands, (FU) provides the programmer with an easy way to scale a numeric input value to the instrument units (Volts, RPM, °K, etc.) of the actual input device. For example, if a tachometer with a range of 0-1000 RPM and a 0 to +5 Volt linear output was connected to an ISAAC 12-bit analog input device, the raw 2048-4095 value returned by the device could be converted, using a function, back into RPM. The formula for this conversion would be:

$$\text{RPM} = (\text{RAW}/2.048) - 1000$$

If this formula had been previously defined as an Applesoft function, it could be used in an & AIN command expression to convert the raw A/D value so that the value stored in (TV) would reflect an actual RPM reading.

The first step in using an optional function with an I/O command is to define the function in normal Applesoft BASIC. For our imaginary tachometer, it would be done like this:

```
250 DEF FN TACH (X) = (RAW%/2.048) - 1000
```

The raw value read by any of LabSoft's input

commands is stored in the reserved variable, RAW%. (Since only the first two characters of a variable name are significant, this variable name may be written as RA%.) Clearly, any conversion function should contain either this reserved variable or the masked raw value reserved variable MASKED% (of which more will be said later).

The second step is to specify the optional function, using the (FU) parameter, in the I/O command line.

```
3010 & AIN, (TV) = A, (FU) = FN TACH (X),(C#) = 0
```

Each time this instruction samples the tachometer-generated voltage through ISAAC's A/D channel #0 the function will be evaluated, and the result (rather than the raw input value) will be stored in the target variable.

The argument variable specified by avar (after the function name when (FU) is used) will substitute for all occurrences of the dummy variable used in the function definition expression. A function equivalent to the one used on the tachometer example could be written.

```
DEF FN TACH (X) = (X/2.048) - 1000
```

Then, a function can be specified in the input command line with an expression of the form

```
(FU) = FN TACH (RAW%)
```

NOTE

The reserved variables, RAW% and MASKED%, are Applesoft integer variables and therefore represent values greater than 32767 as negative equivalents. This will not be a problem when dealing with raw values from devices with 15 bits of accuracy or less. However, this will be a problem if you attempt to use optional functions with devices possessing 16 bits of accuracy. This could happen if you installed, for example, a 16-bit A/D converter in ISAAC, or if you attempted to use an (FU) parameter with a binary input command which considered the 16-bit binary value on the data bus to be one data word. For this reason, we recommend avoiding use of (FU) in binary I/O operations.

Using (FU) With Output Commands

In output commands, (FU) allows the conversion of values used internally by the Apple into values suitable for output to a device. To demonstrate, let's say that ISAAC is in control of a servo that takes an input of -5 to +5 Volts and which can be moved 100 millimeters in increments of one millimeter. To convert a 12-bit D/A output (0 - 4095) into units that correspond with the servo's 100 possible positions, the following formula could be used.

$$\text{OUT} = \text{MM} * 40.95$$

In order to facilitate the use of output functions, LabSoft uses the reserved variable OUT when processing output commands. Each data value to be output is placed in this reserved variable if an (FU) parameter is present in the command line. OUT should be used as part of any output conversion function.

$$190 \text{ DEF FN SERVO (X) = OUT/40.95}$$

Then this optional function would be added to the command line as shown below.

$$5050 \text{ \& AOUT, (DV) = MM, (FU) = FN SERVO (X)}$$

When line 5050 is executed, the specified function will be evaluated, and the resulting value will be output to the hardware.

The argument variable specified by avar (after the function name when (FU) is used) will substitute for all occurrences of the dummy variable used in the function definition expression. Consequently, a function equivalent to the one used above would be:

$$\text{DEF FN SERVO (X) = X * 40.95}$$

Then the function could be specified in the command line with an expression of the form:

$$\text{(FU) = FN SERVO (OUT)}$$

This is the preferred method of using functions in LabSoft commands.

NOTE

If a function is used in any LabSoft I/O command, the function must be interpreted each time a value is input or output. This will decrease execution speed, particularly when a matrix operation is involved. As a rule, the more complex the function, the slower the execution. Therefore, in time-critical I/O operations, it might be wise to avoid optional functions altogether. This does not mean, however, that you must relinquish the use of specific unit conversion. All it means is that you may have to postpone conversion to specific units until after an input operation has been completed, or convert data values prior to an output operation.

- 5.8 (XM) logical eXclusive or Mask
- 5.9 (AM) logical And Mask

(XM) = aexpr
(AM) = aexpr

Aexpr must be an integer in the range of 0-65535 or an ?ILLEGAL QUANTITY ERROR will result. Aexpr should be the decimal equivalent of the desired binary bit mask. If both (AM) and (XM) parameters appear in the same command line, the "AND" masking is always done first, and the result of that operation applied to the "XOR" mask. These masks are optional parameters for most of LabSoft's I/O commands. They are used to mask the raw value read by an input instruction, or to mask data just before it is written by an output instruction. Masks are used most often in binary I/O operations, but they can also be used in analog operations.

Using "AND" Masks

The AND mask has a number of different uses. It is commonly used (among other things) to clear to 0 certain bits of a binary word.

A brief example may help to clarify the workings of the (AM), or logical "AND" mask, parameter. Assume that ISAAC is reading a binary input, and that only the 8 LSB's (Least Significant Bits) of the raw input are of interest. An "AND" mask

with a 1 in bit positions 0-7 (the bits of interest) and a 0 in bit positions 8 - 15 would work as shown in the following table.

	-Binary-	-Decimal-
Raw Input:	1101 0001 1010 1100	53676
"AND" Mask:	0000 0000 1111 1111	255
Result:	0000 0000 1010 1100	172

The "AND" mask above would be specified, using the (AM) parameter, in the following manner:

(AM)=255

The "AND" mask works according to the following truth table. For each bit value in both the data word and the mask word:

1 AND 1 = 1
 1 AND 0 = 0
 0 AND 0 = 0

Remember, we are not adding anything here. What we're doing is satisfying a logical condition; all data bits which are "AND"ed with a mask bit that is 0 will yield a 0.

Using "XOR" Masks

Like the "AND" mask, the "XOR" (logical exclusive OR) mask also has many uses. Among the most common are inverting the state of individual bits for processing purposes, or completely inverting a binary value.

To demonstrate the first application, assume that an ISAAC binary I/O device is set up to monitor 16 valves. The state (open or closed) of each of these valves can be read via a switch on the valve. One side of each switch tied to a TTL-high voltage source. However, eight of the valves have switches that turn on (go "high") when the valve opens; the other eight have switches that turn off (go "low") when the valve opens. Also assume that if any of these valves is open, it will be important to know which one

it is. Rather than curse the idiot who designed such a system, the clever ISAAC user will resort to the (XM) parameter.

If all of the valves are closed, the data will look like this:

	-binary-	-decimal-
Raw Data:	1111 1111 0000 0000	65280

An "XOR" mask can be used on the raw data to invert all of the bits which are "normally on".

	-binary-	-decimal-
Raw Data:	1111 1111 0000 0000	65280
"XOR" Mask:	1111 1111 0000 0000	65280
Result:	0000 0000 0000 0000	0

If the value returned to (TV) is <0 , it will indicate that one of the valves has opened. And it will be easy to tell which valve has opened, since the value returned to (TV) will be 2^n where n = the number of the Binary Input bit to which the open valve is connected.

The "XOR" mask used above would be written using an (XM) parameter in the following manner:

(XM)=65280

The logical "XOR" operation works according to the following truth table:

0	XOR	0	=	0
1	XOR	0	=	1
1	XOR	1	=	0

The (XM) parameter has analog applications as well. To illustrate, assume an ISAAC binary output device is connected to an external 16-bit D/A converter, and an ISAAC binary input device is connected to a 16-bit external A/D converter. This A/D converter accepts a 0-10 Volt input, and returns a proportional 0-65535 binary value. The D/A converter, on the other hand, accepts a 65535-0 input value and returns a proportional 0-10 Volt output. The easiest way to allow your

program to deal with these two converters on the same terms is to use the (XM) parameter to invert the value sent to the D/A converter before it is output.

	-binary-	-decimal-
Data Value:	0000 0000 0000 0001	1
"XOR" Mask:	1111 1111 1111 1111	65535
Output Value:	1111 1111 1111 1110	65534

NOTE The "AND" mask (AM) and the "XOR" mask (XM) can be used together in one LabSoft I/O command. If both are specified in a command, the "AND" mask will execute first.

NOTE Although you will be specifying values in decimal for the masks, they will be converted into 2-byte binary values. Bits more significant than the MSB of your specified value will be considered to have a value of 0. This means that, if you set a mask value equal to 14, using a statement of the form (AM) = 14, the resulting mask will be 0000 0000 0000 1110.

5.10 (GA) Graph Analog flag

(GA)

Most of LabSoft's I/O commands allow you to use this flag in the command expression. When used, it passes the masked raw input or output value associated with the command line on which this flag appears directly to LabSoft's internal graphing routines. If a LabSoft graph has been previously set up, each value input or output will also be automatically graphed. This flag will have no effect if a scrolling or alternating graph has not been previously specified.

In addition to being convenient for the programmer, this flag can save a good deal of execution time. In the following example (next page), the two code segments accomplish the same thing, but the second one executes much faster.

```
1100  & AIN, (TV) = X
1110  & NXTPLT = X/32
```

```
////////////////////////////////////
```

```
5110 & AIN, (TV) = X, (GA)
```

In the first example above, the value plotted by line 1110 is the raw input value divided by 32. This is done because the analog graphing command & NXTPLT accepts an argument in the range of 0-127, while the & AIN command can return a raw value in the range of 0-4095 (assuming it is dealing with a 12-bit converter). If graphing is specified as an optional parameter in an I/O command, the masked raw input or output value is used by the graphing routine, but it is first **automatically** divided down so that it will fall within plottable range (0-127).

- Commands dealing with an analog device have their masked value divided by 32.
- Commands dealing with a binary device have their masked value divided down by 512.
- Commands dealing with a counter device have their masked value divided down by 512.
- Commands dealing with a timer device have their masked value divided down by 512.

NOTE Analog and binary plotting **may not** be done on the same graph. If you're plotting one type and wish to plot the other, you'll have to initialize a new graph.

Caution
********* If your application requires that I/O data be available in its masked raw form, you should avoid using the (GA) parameter and graph that data using a separate & NXTPLT loop instead

5.11 (GB) Graph Binary flag

(GB)

Most of LabSoft's binary I/O commands allow you to specify the (GB) flag. If this graphing flag appears in a LabSoft I/O command expression, the

masked raw value is passed to LabSoft's binary graphing routine. This flag will have no effect if a scrolling or alternating graph has not been previously specified.

5.12 (PR) Print Raw flag

(PR)

When this flag appears in a command line, the masked raw value will be printed to the current display device as an ASCII string followed by a carriage return. This flag is primarily used as a debugging tool (to display unformatted raw incoming values so that they can be checked for accuracy), but it can also be used to display data and to supply raw values for disk storage or serial output to another device.

NOTE The (PR) flag causes the masked raw value to be printed. This value will not reflect the result of any (FU) specified in the same command line.

Caution
***** Although it is possible to include more than one of the display flags (GA), (GB), and (PR) in a command expression, only one can be active at a time. If more than one is specified, only the one which appears last in the expression will be active.

5.13 (RT) sampling RaTe

(RT) = aexpr

Aexpr must be an integer the range 0-16000 or an ?ILLEGAL QUANTITY ERROR will be generated. All of LabSoft's matrix I/O commands allow you to specify a sampling rate. The (RT) parameter specifies the desired sampling rate in milliseconds-per-sample. Note the word "desired." Depending upon which commands and parameter options are used, the minimum execution time for a given command line may be greater than the sampling rate specified by this parameter. If this is the case, the command will execute as fast as it can, without exceeding the specified sampling rate. You will find the chart of typical option execution times in Appendix H of this

manual useful in specifying meaningful sampling rates for various command/option combinations.

NOTE This parameter defaults to a value of 0, (one sample every 0 milliseconds). In other words, unless otherwise specified, samples will be taken as fast as possible. Actual sampling speed in this "free running" mode will depend, as noted above, on the complexity of associated operations.

NOTE Timing of (RT) is derived from the Apple II's system clock, which, as we've noted before, operates at a frequency which will not divide down evenly into seconds (or milliseconds). The actual value for each unit specified by the argument of (RT) will be approximately 1.00343 milliseconds.

5.14 (SW) number of I/O SWeeps

(SW) = aexpr

Aexpr must be in the range of 1-65535 or an ?ILLEGAL QUANTITY ERROR will be generated. This parameter specifies the number of times the associated matrix command will be executed. (SW) is particularly useful for specifying the number of times an output pattern is to be repeated, or for determining the number of samples to sum in an analog sum (& ASUM) operation.

NOTE This parameter has a default value of 1. Matrix commands which do not include an (SW) parameter in the command line will execute once for each element of (AV). Specifying a value of 0 for (SW) will generate an ?ILLEGAL QUANTITY ERROR.

5.15 (CV) Compare Value

(CV) = aexpr

Aexpr can be any valid Applesoft real arithmetic expression. All of LabSoft's input commands allow you to specify (CV) in the command expression. This can be used to perform an automatic comparison of an input value with the value in

the argument of (CV). The result of the comparison can be found by reading the state of the reserved variables LT%, EQ%, and GT% according to the following table.

(CV) Truth Table

```

-----
If (TV)  <  (CV)  Then LT% = 1, EQ% = 0, GT% = 0
If (TV)  =  (CV)  Then LT% = 0, EQ% = 1, GT% = 0
If (TV)  >  (CV)  Then LT% = 0, EQ% = 0, GT% = 1

```

NOTE

LT%, EQ%, and GT% are essentially "flags", which are "set" (equal to 1) to indicate one of three possible true conditions. In simple input operations, LT%, EQ%, and GT% will be set as shown above. However, in matrix operations, the state of the flags will be set for each array element but not cleared. This means that in matrix operations, the state of the flags will indicate that at least one of the values input by the matrix operation has met the indicated condition. If EQ% = 1 after a matrix input command, at least one of the values read during the execution of that command was equal to the aexpr in the argument of (CV).

**Caution

Each execution of an I/O operation will reset LT%, EQ%, and GT% appropriately. Beeper/Buzzer commands (output commands) will reset all three to 0.

5.16

(W#) Word #

(W#) = aexpr

Aexpr must be an integer in the range 0-15 or an ?ILLEGAL QUANTITY ERROR will result. This parameter is required (and only used by) the expansion commands & WRDEV and & RDEV. It specifies the binary word to which data is written or from which data is read. This parameter will be explained in greater detail in Appendix X of this manual.

A Note on Matrix and Triggered Operations

Now that you know your way around LabSoft's parameters, we're going to take a little time to introduce you to the concepts of Matrix and Triggered operations.

Matrix Operations

Matrix (or array) commands allow you to perform a specified I/O operation for each element of a specified array. This structure is particularly useful in programs designed for data acquisition. In operation, matrix or array commands are as easy to use as simple commands.

A simple analog input fetches one analog data value from ISAAC and stores it in a target variable (TV) specified by the programmer. A matrix analog input inputs as many analog values from ISAAC as there are elements in an array variable (AV) specified by the programmer. The first input value is stored in the first element of the array. The second input value will be stored in the second element of the array. This process continues until all elements of the array are filled.

Not only does this structure allow the program to input or output data to and from arrays in one simple command, it also allows the reading or writing of data in evenly timed periods (using the (RT) and (SW) parameters), at speeds potentially much faster than those that could be attained using FOR...NEXT loops written in Applesoft BASIC.

Nearly all of LabSoft's I/O commands can be matrixed. The only ones that cannot would probably offer no advantages to the programmer if they could be. The "at" sign is used (as in &@AIN) to indicate to LabSoft that the command is a matrix (or @rray) command, in this case, a matrix analog input.

NOTE

When using the matrix commands, you may occasionally notice a delay between the initial interpretation of a command and its execution. You may also notice that this occurs only when

your command is used with a very large array and when LabSoft's reserved variables (RAW%, MASKED%, OUT, LT%, EQ%, and GT%) have not yet appeared in the program. This happens because of the way simple variables and arrays are stored in memory by BASIC. This delay can be easily avoided, however, by using any of the following programming techniques:

- A. "Declare" all of the reserved variables (and any other simple variables that you plan on using in command expressions or optional functions). Although BASIC doesn't have declaration statements, a line like the one that follows will do nicely.

```
100 RAW% = 0 :MASKED% = 0 :OUT = 0 :  
    LT% = 0: EQ% = 0 : GT% = 0
```

- B. Execute any I/O command. A simple input command, like the one shown below, will work.

```
100 & AIN,(C#) = 1,(TV) = X
```

Remember that these techniques, if they're going to work, must be done **before** any arrays are dimensioned. That way, they will be able to inform BASIC of reserved variable placement and will speed subsequent interpretation.

Triggered Operations

In many data acquisition applications, it is often necessary to trigger an I/O operation when a particular event occurs. You may need to check a furnace temperature each time a new batch of material is put into it, or record a laboratory animal's heart rate whenever it pushes a certain lever. Of course, you could program these operations in BASIC using LabSoft and the IF...THEN... structure allowed in AppleSoft BASIC. However, if the time interval between the triggering event and the triggered operation must be kept short, a BASIC structure might not be appropriate. It depends on the complexity of the two operations. In many cases, the time required to confirm the occurrence of a valid triggering event and then trig-

ger the subsequent operation could be anywhere from three to hundreds of milliseconds. If that were the case, much of the incoming data might easily be missed.

LabSoft's & LOOK FOR...THEN... command structure allows you to use most of LabSoft's I/O commands as trigger commands for other commands. A threshold value (the value to "LOOK FOR") is set using the (TH) parameter. When an expression of this type is executed, the data value of the trigger command is tested continuously until the specified threshold condition is met. The THEN part of the expression (which can be any valid Applesoft or LabSoft command) is then executed in a matter of microseconds, which should be fast enough for most applications.

The & LOOK FOR...THEN... structure is unique and flexible. We've devoted an entire chapter of this manual (Chapter 10) to its various uses.

I/O Sequences and Reserved Variables

Partly because of the nature of LabSoft, and partly because of the way the parameters are used, many things happen during the execution of a typical I/O command. The order in which these things happen is important for the programmer to know, particularly when considering the contents of LabSoft's reserved variables, and when using functions or masks. The sequence of events is different for input operations than it is for output operations.

For each occurrence of a **simple input** command, or each iteration of a **matrix input** command:

1. Data is input from hardware as a 2-byte binary word and stored in the reserved variable RAW%.
2. If an (AM) or (XM) mask has been specified, the input is masked and the result stored in the reserved variable MASKED%.
3. The optional function (FU) is evaluated.
4. The optional comparison (CV) is performed.

5. The result of the most recent operation is stored in the specified Target Variable (TV) or, if the command is a matrix command, in the next element of the Array Variable (AV).
6. If a graphing flag (GA) or (GB) is present, the value of the reserved variable MASKED% is graphed. If there is a (PR) flag present, the value of MASKED% will be displayed as an unsigned ASCII string (with a carriage return). Only the last of these three flags to appear in a given command expression will be executed. Any others will be ignored.

NOTE

Any additional conversions (BCD, frequency, etc.) will usually be done between steps 1 and 2. The specific command descriptions in later chapters will provide more detail on this.

For each occurrence of a **simple output** command, or each iteration of a **matrix output** command:

1. The Data Value (DV) or the current array element value of Array Variable (AV) is stored in the reserved variable OUT. If an optional function (FU) is present, it is evaluated.
2. This value is converted to a 2-byte unsigned integer and stored in the reserved variable RAW%.
3. If an (AM) or (XM) mask is present, the reserved variable RAW% is masked and the result is stored in the variable MASKED%.
4. The value in MASKED% is output to ISAAC as an unsigned 2-byte integer.

NOTE

There is a table of LabSoft's reserved variables in Appendix B of this manual.

We hope that by now you're beginning to get a feel for the way LabSoft operates. By the time you've read the next few chapters, you should know just about all there is to know about this language. Remember, you don't have to read every word. Every other word would probably be enough for openers. From here on out, you can, if you wish, concentrate on the parts that are

of particular interest to you. Of course, if you do want to read the whole thing, we're not going to stand in your way.

&&&



CHAPTER 6

THE ANALOG I/O COMMANDS

- 6.1 & AIN
& @AIN
- 6.2 & ASUM
& @ASUM
- 6.3 & AOUT
& @AOUT
- 6.4 & ANAFMT

CHAPTER 6: THE ANALOG I/O COMMANDS

Electrical signals which vary continuously over a fixed range are called **analog** signals. Variations in these signals typically represent (are analogous to) changes in some real-world quantity. ISAAC analog input systems measure these signals and convert them to digital values suitable for storage, processing, transmission, or display. ISAAC analog output systems convert digital values to corresponding analog signals which can then be output to the real world.

LabSoft's analog I/O commands include instructions to input, input and sum, and output analog values in user-specified patterns. These commands access the following ISAAC analog I/O devices.

Default Analog Device, ISAAC 91A

I-100 16 Channel 12-Bit A/D Converter

I-110 4 Channel 12-Bit D/A Converter

I-130/I-130A Preamp Control Modules

In this chapter, the analog I/O commands will be listed along with their required parameters. A detailed description of the command's function and a list of optional parameters will follow. The matrix versions of these commands (identifiable by the @ symbol preceding the command word) will be covered in the discussion of the simple versions.

NOTE

As we mentioned in the previous chapter, matrix commands execute a given operation for a specified number of times at a specified rate. Simple commands execute a given operation once.

6.1

& AIN
& @AIN

& AIN, (TV) = avar
& @AIN, (AV) = numaryname

These commands read (input) an analog value from an analog input device in an ISAAC system. Where no default device exists, a (D#) (device number) parameter must be included in the command line. If a (D#) parameter is not specified, LabSoft will not know which analog input device to access. If a (C#) (channel number) parameter is included in the command line, that channel of the specified device will be used. Otherwise, the "next channel" from the most recently specified analog channel format (& ANAFMT) command will be used.

Caution If the device location specified by (D#) is not present, or is not occupied by a suitable analog input device, the system may hang or crash.

NOTE The model 91A ISAAC has a default analog device. & AIN and & @AIN will access this device when no (D#) parameter is included in the command line.

The following parameters are optional with & AIN:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with & @AIN:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =
		Sampling RaTe	(RT) =
		No. of SWeeps	(SW) =

6.2

& ASUM
& @ASUM

& ASUM, (TV) = avar
& @ASUM, (AV) = numaryname

Like **& AIN**, these commands also read an analog value from an ISAAC analog device. However, instead of simply storing that value in (TV) or (AV), these commands **sum** that value with the other values in the specified target variable (TV) or array (AV) element. Channel (C#) and Device (D#) requirements are the same as for **& AIN**.

Caution
********* Since this command performs repetitive arithmetic operations, it is important to be aware that arithmetic overflows can occur. This is particularly true when using integer variables and arrays. If an arithmetic overflow occurs, an **?ILLEGAL QUANTITY ERROR** or **?OVERFLOW ERROR** will be generated, and program execution will halt.

Caution
********* If the device location specified by (D#) is not present, or is not occupied by a suitable analog input device, the system may hang or crash.

The following parameters are optional with both **& ASUM** and **& @ASUM**.

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =
		Sampling RaTe	(RT) =
		No. of SWeeps	(SW) =

Analog Signal Averaging Using **& ASUM**

Software signal averaging can be a valuable technique for reduction of random noise present on an analog signal. There are several types of signal averaging, of which block, (or "boxcar") averaging is one. LabSoft's **& ASUM** command makes it easy to boxcar-average analog input for noise reduction. The following program demonstrates how **& ASUM** may be used to produce an

array of 60 signal-averaged data points, each representing the average of 100 samples of 10 milliseconds duration each (a rate of 100 samples per second).

```
100 DIM D (59)
110 FOR N = 0 to 59
120 & ASUM, (TV) = X, (RT) = 10, (SW) = 100
130 D (N) = X/100
140 NEXT N
```

Line 100 DIMensions the data array to hold 60 data points. Line 120 collects the **sum** of 100 samples of .01 second each. And line 130 divides that value by 100, placing the average of those 100 readings in array D.

NOTE

& ASUM does not clear the target or array variable first. To ensure that new data acquired by & ASUM isn't summed with previously existing data, clear the array first, using LabSoft's & ARRAYCLR command. It's also worth noting that the number of **periods** required for an execution of & ASUM (or **any** matrix execution) will always be **one less** than the specified number of sweeps (SW). This is because the first sweep is always executed immediately. In the example above, line 120 takes 990 milliseconds to execute.

NOTE

& ASUM is the only simple command that allows you to specify (RT) and (SW).

6.3

& AOUT
& @AOUT

& AOUT, (DV) = aexpr
& @AOUT, (AV) = numaryname

These commands write (output) a value to an analog output device in an ISAAC system. Where no default device exists, a (D#) (device number) parameter must be included in the command line. If a (D#) parameter is not specified, LabSoft will not know which analog output device to access. If a (C#) (channel number) parameter is included in the command line, that channel of the specified device will be used. Otherwise, the "next channel" from the most recently specified & ANAFMT command will be used.

Caution If the device location specified by (D#) is not
***** present, or is not occupied by a suitable analog
 output device, the system may hang or crash.

NOTE The model 91A ISAAC has a default analog device.
 & AOUT and & @AOUT will access this device when
 no (D#) parameter is included in the command
 line.

If an analog output command line specifies an output value greater than the maximum value readable by the D/A converter in the device being accessed, the extra significance will "wrap around". All ISAAC analog output devices contain 12-bit D/A converters. Any value greater than 4095 written to such devices will be interpreted as that value MOD 4095 (e.g., 4096 = 0, 4097 = 1, etc.)

The following parameters are optional with & AOUT:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with & @AOUT:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		— Compare Value	(CV) =
		— Sampling RaTe	(RT) =
		— No. of SWeeps	(SW) =

& ANAFMT

& ANAFMT = aexpr1 [,aexpr2..., aexprN]

Aexpr must be an integer in the range 0-255 or an ?ILLEGAL QUANTITY ERROR will result. Attempts to specify more than 16 arguments (aexpr) will result in a ?SYNTAX ERROR.

The arguments of **& ANAFMT** specify which channels LabSoft will access when it encounters an analog I/O command (**& AIN**, **& ASUM** and **& AOUT**). The order in which these arguments are stated specifies the order in which the channels will be accessed. Where a (C#) (channel number) parameter is included in an analog I/O command line, that channel of the specified device will be used. Otherwise, the "next channel" from the most recently specified **& ANAFMT** command will be used.

Each time LabSoft encounters an analog I/O command line which doesn't include a (C#) parameter, it assumes that the "next channel" in the currently active **& ANAFMT** is the channel to use. After each such use, the "next channel" pointer is incremented (advanced to the next argument). Analog I/O command lines which include a (C#) parameter will access the channel specified by that (C#) and will **not** affect the "next channel" pointer.

NOTE The "next channel" pointer is incremented for **each iteration** of a matrix command where no (C#) is included in the command line.

NOTE Re-executing the **& ANAFMT** (or executing an **& FMTDFLT**) command is the only way to explicitly reset the "next channel" to the first one in a specified **& ANAFMT** statement.

The arguments of **& ANAFMT** are bound when the **& ANAFMT** command is executed, **not** when the analog I/O commands that use them are executed. Notice also that a specified channel number must not be greater than the maximum value allowed for the associated analog I/O command. If it is, the channel number will "wrap around" as we mentioned earlier.

How Sampled Data Are Stored in an Array

When using data acquired by ISAAC, it's important to know the source (i.e., the input channel) of each point of data in a data array. This can be somewhat tricky when you're filling multidimensional arrays with data acquired during matrix operations. To fill an array with ten samples from each of four analog channels, you could take ten samples from the first channel, ten from the second, and so on, or you could use & ANAFMT as shown here:

```
100 DIM AN(3,9)
```

```
////////////////////////////////
```

```
5000 & ANAFMT = 1,3,5,2
```

```
5010 & @AIN, (AV) = AN
```

Line #5000 specifies that channels numbered 1,3,5, and 2 should be sampled in that order. Line #5010 fills the array with analog values. For each iteration of the & @AIN command, the analog value will be taken from the "next channel". After the program above has been executed, the contents of array AN will bear the following relationship to the sampled channels:

ARRAY ELEMENT	SAMPLE FROM CHANNEL #
---------------	-----------------------

AN(0,0)	1
AN(1,0)	3
AN(2,0)	5
AN(3,0)	2

```
////////////////////////////////
```

AN(1,9)	3
AN(2,9)	5
AN(3,9)	2

The "circular" nature of the analog channel format should be evident from the table above. Any array element (0,n) will come from the first channel of the format, element (1,n) will come from the second channel of the format, and so on through the entire array.

&&&

This Page Intentionally Left Blank



CHAPTER 7

THE BINARY I/O COMMANDS THE SCHMITT TRIGGER COMMANDS

- 7.1 & BIN
& @BIN
- 7.2 & BOUT
& @BOUT
- 7.3 & BCDIN
& @BCDIN
- 7.4 & BCDOUT
& @BCDOUT
- 7.5 & BPOLL
& @BPOLL
- 7.6 & TRIGIN
& @TRIGIN
- 7.7 & TRIGPOLL
& @TRIGPOLL



CHAPTER 7: THE BINARY I/O COMMANDS; THE SCHMITT TRIGGER COMMANDS

Unlike analog signals, which can assume an infinite number of values within a given range, binary signals can assume only two values: high or low. Like most other computer systems, ISAAC interprets binary signals as the "bits" (1s and 0s) which make up binary numbers. ISAAC binary devices read and write a binary data "word" 16 bits wide. The bits of this word may be interpreted as:

- an integer in the range 0-65535
- a BCD (Binary Coded Decimal) integer in the range 0-9999
- a series of unrelated bits

LabSoft's binary I/O commands include instructions which read and write binary values, read and write BCD values, and poll binary inputs. These commands access the following ISAAC binary I/O devices:

Default Binary Device, ISAAC 91A

I-120 16-bit Binary I/O Module

In order to fully understand the way in which these commands work, it will be necessary for you to have some knowledge of the binary number system. This subject has been dealt with in detail in more than a few publications. We recommend that you seek out one of these if you need additional information on this subject.

NOTE

There are four Schmitt triggers included in the default binary device of the ISAAC 91A. LabSoft's Schmitt trigger commands read Schmitt trigger states as binary values, which is why these commands are included in this section.

7.1

& BIN
& @BIN

& BIN, (TV) = avar
& @BIN, (AV) = numaryname

These commands read (input) the **decimal value** of the 16-bit binary input word of an ISAAC binary device. Where no default binary device exists, a (D#) (device number) parameter must be included in the command line. If no (D#) parameter is specified, LabSoft will not know which binary device to access.

If the location specified by (D#) is not present, or is not occupied by a binary device, these commands will be executed, but the values returned will be meaningless.

NOTE

The model 91A ISAAC has a default binary device. **& BIN** and **& @BIN** will access this device when no (D#) parameter is included in the command line.

The following parameters are optional with **& BIN**:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with **& @BIN**:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =
		Sampling RaTe	(RT) =
		No. of SWeeps	(SW) =

NOTE

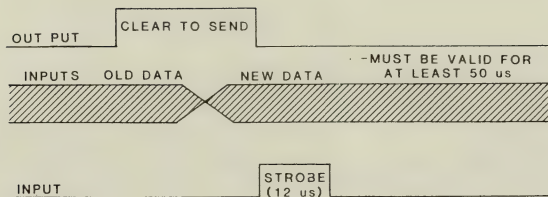
Numeric values greater than 32767, when stored in Applesoft integer form (i.e., in variables flagged with a % symbol), are represented by Applesoft as their negative equivalents. Since binary input commands read values to 65535,

these negative equivalents may pose a problem when integer variables are used for (TV) or (AV). To eliminate this problem, use real variables for input values greater than 32767, or (for values with fewer than 16 significant bits) mask out insignificant high-order bits on input.

Binary Input Handshaking

LabSoft's binary input commands support communication handshaking through ISAAC binary device's CLEAR TO SEND and STROBE lines. These lines are normally pulled up to their "clear-to-send" state. You won't have to make any connections to these lines unless input handshaking is required. When input handshaking is used, each execution of a simple binary input command or iteration of a matrix binary input command follows the protocol below. (Fig. 7:1, next page)

1. ISAAC sets the binary input port's CLEAR TO SEND line high. This indicates to an external device that ISAAC is ready to receive new data.
2. As soon as the data is placed on the data lines, either of the binary input port's STROBE lines should be strobed (set high for at least 12 microseconds, then brought low again) by the external device. Data on the binary data lines must be stable for at least 50 microseconds from the leading edge of the STROBE pulse.
3. The binary input port's CLEAR TO SEND line is brought low when the input data STROBE is detected. This indicates that ISAAC has received the data and is not yet ready for the next word.



-FIGURE 7:1-

This handshaking protocol will only be required when data is being transferred at exceptionally high rates between ISAAC and another intelligent device. Input handshaking is supported by the following LabSoft commands.

& BIN & @BIN
 & BCDIN & @BCDIN
 & TRIGIN & @TRIGIN

NOTE When a **simple** binary input command must wait for a STROBE from an external device, program execution will halt until that STROBE is received. When a **matrix** binary input command must wait for a STROBE from an external device, the command will execute at less than the specified rate if that rate is faster than the transmitting rate of the external device. In such cases, the command will sample as fast as it can.

NOTE ISAAC binary inputs are standard LS TTL and are not conditioned in any way. Any required signal conditioning (like mechanical switch de-bounce) will have to be done by external circuitry.

7.2

& BOUT
& @BOUT

& BOUT, (DV) = aexpr
& @BOUT, (AV) = numaryname

These commands write (output) a **decimal value** as a 16-bit binary word via the output port of an ISAAC binary device. Where no default device exists, a (D#) (device number) parameter must be included in the command line. If a (D#) parameter is not specified, LabSoft will not know which binary device to access.

NOTE

The model 91A ISAAC has a default binary device. & BOUT and & @BOUT will access this device when no (D#) parameter is included in the command line.

The following parameters are optional with & BOUT:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with & @BOUT:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =
		Sampling RaTe	(RT) =
		No. of SWeeps	(SW) =

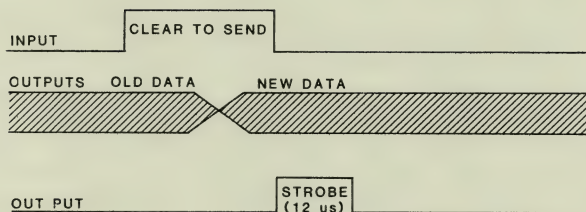
NOTE

Since the binary output port of an ISAAC binary device is 16 bits wide, the final output value specified by & BOUT or & @BOUT must be in the range of 0-65535 (-32767 to +32767 if Applesoft integer variables are being used), or an ?ILLEGAL QUANTITY ERROR will be generated and program execution will halt.

Binary Output Handshaking

LabSoft's binary output commands support communication handshaking via the binary output port's CLEAR TO SEND and STROBE lines. These lines are normally pulled up to their "clear-to-send" state. You won't have to make any connections to these lines unless input handshaking is required. Each execution of a simple binary output command and each iteration of a matrix binary output command follows the protocol listed below. (Fig. 7:2, next page)

1. The external device (to which data is to be sent) indicates a "not ready to receive" condition by pulling the CLEAR TO SEND line low. Execution of & BOUT or & @BOUT pauses until the CLEAR TO SEND line is set high. (This line is normally held high by an internal pull-up resistor.)
2. The specified 16-bit data value is put on the binary output port's data lines.
3. ISAAC sets the output STROBE line high for 10 microseconds to indicate to the external device that there is valid data on the bus. It then returns the STROBE line low.
4. The external device completes the protocol by pulling the CLEAR TO SEND line low. This indicates to ISAAC that the device has received the data and is not yet ready for more.



-FIGURE 7:2-

Communications handshaking will only be required when data is being transferred at high rates between ISAAC and another intelligent device. Output handshaking is supported by the following LabSoft commands.

& BOUT & @BOUT
 & BCDOUT & @BCDOUT

NOTE

When a **simple** binary output command must wait for a CLEAR TO SEND signal from an external device, program execution will halt until that signal is received. When a **matrix** binary output command must wait for a CLEAR TO SEND signal from an external device, the command will execute more slowly than the specified rate if that rate is faster than the transmitting rate of the external device. In all such cases, the command will execute as fast as it can.

7.3

& BCDIN
& @BCDIN

& BCDIN, (TV) = avar
& @BCDIN, (AV) = numaryname

These commands read (input) a 16-bit binary word from the input port of an ISAAC binary device and evaluate that word as a four digit BCD (Binary Coded Decimal) value. Where no default device exists, a (D#) (device numbe) parameter must be included in the command line. If a (D#) parameter is not specified, LabSoft will not know which binary device to access. These commands support input handshaking as discussed in section 7.1 of this chapter.

NOTE

The model 91A ISAAC has a default binary device. & BCDIN and & @BCDIN will access this device when no (D#) parameter is included in the command line.

The following parameters are optional with & BCDIN:

Analog Graph Flag	(GA)	Device #	(D#) =
		FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with & @BCDIN:

Analog Graph Flag	(GA)	Device #	(D#) =
		FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =
		Sampling RaTe	(RT) =
		No. of SWeeps	(SW) =

Caution

Illegal BCD values (i.e., digit values that are greater than 9) will yield an ?ILLEGAL QUANTITY ERROR on input and program execution will be halted. If the device location specified by

suitable binary device, it is likely that illegal BCD values will be returned. For the same reason, any input lines on which no data will be present should be either masked out with an (AM) parameter or tied low (with a jumper to logic ground).

How LabSoft Interprets BCD

& BCDIN and & @BCDIN first input and mask the binary input word, then store the BCD interpretation of the masked value of that word in (TV) or (AV). Since the conversion to BCD occurs after the binary value has been input and masked, the reserved variables RAW% and MASKED% will contain the decimal (rather than the BCD) equivalent of the input word. For most BCD applications, this value won't be particularly useful. It means, however, that you will have to specify mask values for BCD words in the same way that you would for a binary ones.

LabSoft interprets binary values as unsigned BCD integers as shown in the table below.

BIT#	15	14	13	12	/	11	10	9	8	/	7	6	5	4	/	3	2	1	0

BCD digit	1000's				/	100's				/	10's				/	1's			

An "AND" mask to mask out the 100's and 1000's digits of a BCD value would be specified

(AM) = 255

and would operate as shown below.

Input Word =	1000	0010	0000	1001
"AND" mask	0000	0000	1111	1111
MASKED% =	0000	0000	0000	1001
BCD value =	0	0	0	9

7.4

& BCDOUT
& @BCDOUT

&BCDOUT, (TV) = aexpr
& @BCDOUT, (AV) = numaryname

These commands write (output) a decimal value as unsigned four digit BCD to the output port of an ISAAC binary device. Where no default device exists, a (D#) (device numbe) parameter must be included in the command line. If a (D#) parameter is not specified, LabSoft will not know which binary device to access. These commands support output handshaking as discussed in section 7.2 of this chapter.

NOTE

The model 91A ISAAC has a default binary device. **& BCDOUT** and **& @BCDOUT** will access this device when no (D#) parameter is included in the command line.

The following parameters are optional with **& BCDOUT**:

Analog Graph Flag	(GA)	Device #	(D#) =
		FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with **& @BCDOUT**:

Analog Graph Flag	(GA)	Device #	(D#) =
		FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =
		Sampling RaTe	(RT) =
		No. of SWeeps	(SW) =

Caution *****

The final output value written by these commands must be an integer in the range 0-9999 or an **?ILLEGAL QUANTITY ERROR** will be generated and program execution will be halted.

7.5

& BPOLL
& @BPOLL

& BPOLL, (TV) = avar
& @BPOLL,(AV) = numaryname

These commands continuously read and mask the value at the input port of an ISAAC binary device. The individual bits of the masked input value are examined, starting with bit #0 and progressing through bit #15. This process continues until one of the bits of the input word is found to be "set" (i.e., equal to 1). The number (0-15) of the lowest-order "set" bit will be returned to (TV) or to the designated array element of (AV). Where no default device exists, a (D#) (device number) parameter must be included in the command line. If a (D#) parameter is not specified, LabSoft will not know which binary device to access.

If the device location specified by (D#) is not present, or is not occupied by a suitable binary device, these commands will execute but the value returned will be meaningless (probably 0).

NOTE

The model 91A ISAAC has a default binary device. & BIN and & @BIN will access this device when no (D#) parameter is included in the command line.

The following parameters are optional with & BPOLL:

	Device #	(D#) =
	FUnction	(FU) =
Print Raw Value Flag (PR)	"AND" Mask	(AM) =
	"XOR" Mask	(XM) =
	Compare Value	(CV) =

The following parameters are optional with & @BPOLL:

	Device #	(D#) =
	FUnction	(FU) =
Print Raw Value Flag (PR)	"AND" Mask	(AM) =
	"XOR" Mask	(XM) =
	Compare Value	(CV) =
	sampling RaTe	(RT) =
	no. of SWeeps	(SW) =

NOTE

If two or more of the binary input lines go high at once, only the number of the lowest order line will be returned. The full binary input value read during the polling process will be contained in the reserved variable MASKED%. Among other things, this command is particularly useful as a trigger in LabSoft's & LOOK FOR... THEN... command, which will be dealt with at length in Chapter 10 of this manual.

NOTE

Execution of this instruction is dictated by the state of the bits of the binary input port. Since the system may pause for an indefinite amount of time waiting for a true (high bit somewhere) condition, & @BPOLL may not execute at the specified rate, and both & @BPOLL and & BPOLL may appear at times to hang. If, for some reason, you can't wait, you may use CTRL-C to halt execution.

The Schmitt Trigger Commands

A Schmitt trigger is a device that accepts analog input and provides a binary output based on the level of input voltage with respect to a preset threshold voltage. In the ISAAC 91A, each of the four Schmitt triggers in the Schmitt trigger subsystem will be set and latched when the voltage input to that trigger exceeds the preset threshold voltage. The state of the Schmitt trigger subsystem of the ISAAC 91A may be read as the decimal equivalent of a 4-bit binary value. A set trigger will return a 1 in its' bit position. A cleared trigger will return a 0.

These commands access the Schmitt trigger subsystem of the default binary device of the ISAAC 91A. Any use of these commands to access a binary device which contains no Schmitt triggers will return the value of binary bits 0-3.

7.6

& TRIGIN & @TRIGIN

```

& TRIGIN, (TV) = aexpr
& @TRIGIN, (AV) = numaryname

```

These commands read the states of the four Schmitt triggers as a binary word, which we'll refer to as the Schmitt word. These commands return an unmasked value in the range 0-15 from the Schmitt trigger subsystem of the ISAAC 91A. These commands support input handshaking as described in section 7.1 of this chapter. Data input to the Schmitt triggers should be in **ana-log** (rather than binary) form.

If the device location specified by (D#) is not present, or is not occupied by a suitable binary device, these commands will execute, but the value(s) will be meaningless.

The following parameters are optional with & TRIGIN:

	Device #	(D#) =
Binary Graph Flag	(GB) FUnction	(FU) =
Print Raw Value Flag	(PR) "AND" Mask	(AM) =
	"XOR" Mask	(XM) =
	Compare Value	(CV) =

The following parameters are optional with & @TRIGIN:

	Device #	(D#) =
Binary Graph Flag	(GB) FUnction	(FU) =
Print Raw Value Flag	(PR) "AND" Mask	(AM) =
	"XOR" Mask	(XM) =
	Compare Value	(CV) =
	No. of SWeeps	(SW) =
	Sampling RaTe	(RT) =

How LabSoft Reads Schmitt Trigger States

The value returned by these commands correlates with the states of the four triggers according to the following table.

Unmasked Value Decimal	Binary	Trigger #s Set
0	0000	0
1	0001	1
2	0010	2
3	0011	1,2
4	0100	3
5	0101	1,3
6	0110	2,3
7	0111	1,2,3
8	1000	4
9	1001	1,4
10	1010	2,4
11	1011	1,2,4
12	1100	3,4
13	1101	1,3,4
14	1110	2,3,4
15	1111	1,2,3,4

Each Schmitt trigger will be set high and latched when the signal input to the trigger exceeds the threshold voltage to which that trigger has been set. The trigger will remain latched until the Schmitt word is read by an & TRIGIN or & @TRIGIN command, which clears the latch and allows the triggers to reset where the input signal has fallen below the threshold.

NOTE

The Schmitt triggers are always read-enabled. Any input signal equal to or greater than a trigger's established threshold will cause the trigger to be set and latched. Since the only way to clear the latch on the ISAAC 91A's Schmitt trigger subsystem is to read the state of the triggers (using & TRIGIN), it is best to issue a "dummy" & TRIGIN command (one where the value of (TV) is presumed to be meaningless) early in any program involving this subsystem. That way, you can be certain that the triggers were unlatched at a certain point in the program, and that subsequent changes-of-state represent valid responses to known inputs.

7.7

& TRIGPOLL
& @TRIGPOLL

```
& TRIGPOLL (TV) = aexpr
& @TRIGPOLL, (AV) = numaryname
```

These commands continuously read and mask the value at the Schmitt trigger port of the ISAAC model 91A. The individual bits of the masked input value are examined, starting with bit #0 and progressing through bit #3. This process continues until one of the bits of the Schmitt word is found to be "set" (i.e., equal to 1). The number (0-3) of the lowest-order "set" bit will be returned to (TV) or to the designated array element of (AV).

If the device location specified by (D#) is not present, or is not occupied by a Schmitt trigger device, these commands will execute, but the value(s) returned will be meaningless.

The following parameters are optional with & TRIGPOLL:

	Device #	(D#) =
	FUnction	(FU) =
Print Raw Value Flag (PR)	"AND" Mask	(AM) =
	"XOR" Mask	(XM) =
	Compare Value	(CV) =

The following parameters are optional with & @TRIGPOLL:

	Device #	(D#) =
	FUnction	(FU) =
Print Raw Value Flag (PR)	"AND" Mask	(AM) =
	"XOR" Mask	(XM) =
	Compare Value	(CV) =
	sampling RaTe	(RT) =
	no. of SWeeps	(SW) =

NOTE

If two or more of the binary input lines go high at once, only the number of the lowest order line will be returned. The full binary input value read during the polling process will be contained in the reserved variable MASKED%. Like their & BPOLL counterparts, either of these commands can be very useful as a trigger in LabSoft's & LOOK FOR... THEN... command, which

will be dealt with at length in Chapter 10 of this manual.

NOTE

Execution of this instruction is dictated by the state of the bits of the Schmitt word. Since the system may pause for an indefinite amount of time waiting for a true (high bit somewhere) condition, & @TRIGPOLL may not execute at the specified rate, and both & @TRIGPOLL and & TRIGPOLL may appear at times to hang. If, for some reason, you can't wait, you may use CTRL-C to halt execution.

&&&



CHAPTER 8

THE COUNTER COMMANDS

- 8.1 & CLRCOUNTER
- 8.2 & CNTFMT
- 8.3 & COUNTERIN
- 8.4 & FINL
& @FINL
- 8.5 & FINH
& @FINH

CHAPTER 8: THE COUNTER COMMANDS

The model 91A ISAAC contains a 16-bit externally-clocked count-up Counter Device, device number 14. This default counter has eight multiplexed input channels. Seven of these channels accept TTL input and can count through a frequency range of 0 (DC) to 10 megaHertz. The remaining channel (channel #7) is conditioned to accept non-TTL signals at levels above 100 millivolts (RMS) in a frequency range of 0.5 Hertz to 3 megaHertz. In addition to providing simple counter access, LabSoft has commands that will interpret the value of the count as a frequency.

NOTE

On the ISAAC 91A, device #14 is also the residence of the Beeper/Buzzer hardware discussed in Chapter 3 of this manual. The Beeper/Buzzer and the Counter are in no way related but, as is common these days, just live together.

These commands will only access the default counter device of the ISAAC 91A

8.1

& CLRCOUNTER

& CLRCOUNTER

This command clears the value of the default (or otherwise specified) Counter Device's count-up counter to zero. An optional (C#) parameter may be used with this command to specify which of the inputs you wish to count after the counter has been cleared. Since there is only one counter, only one count can be taken at a given time. From the time an & CLRCOUNTER is executed, every cycle of the signal at the designated counter input will increment the count by one. If you don't specify a channel number, & CLRCOUNTER will use the channel specified by the "next channel" of the most recently executed & CNTFMT. See Section 8.2 for more on this.

The following parameters are optional with & CLRCOUNTER:

Device # (D#) = Channel # (C#) =

8.2

& CNTFMT

& CNTFMT = aexpr1 [, aexpr2..., aexprN]

The argument(s) of & CNTFMT correspond to the counter channel(s) from which LabSoft will read counts. The order in which these arguments are written is the order in which the channels will be accessed in successive counting operations. The model 91A ISAAC's default counter has an 8-channel multiplexer, so aexpr should be an integer in the range of 0-7. If a value greater than 7 is used as an argument of & CNTFMT, that value will "wrap around" (the actual counter channel used will be aexpr MOD 8).

NOTE

The maximum number of arguments allowed with & CNTFMT is 16. These arguments are bound at the time the command is executed. If more than 16 arguments are used in an & CNTFMT command line, a ?SYNTAX ERROR will be generated when that command line is executed.

NOTE

Although each execution of & CLRCOUNTER, & FINL, or & FINH (as well as each iteration of

& @FINL, or & @FINH) will increment the "next channel pointer", only the execution of an & CNTFMT or & FMTDFLT command can explicitly reset the "next channel pointer" to the first argument of & CNTFMT.

Counter Channel Formatting

Like all LabSoft commands dealing with I/O formatting, & CNTFMT allows you to specify a channel format to which subsequent I/O commands of that type will refer. Whenever LabSoft executes an & CLRCOUNTER command line which doesn't contain a (D#) parameter, it will access the "next channel" from the currently active & CNTFMT. The counter device, remember, can also be read specifically as a frequency counter, and & CNTFMT applies to this situation as well. A brief example will help to make the function of this command clear.

Assume that each of three counter channels is connected to a speed-sensing device on one of three high-speed motors. If this device sends one TTL pulse per revolution, the following program would allow a model 91A ISAAC to read the speed of each motor in revolutions-per-second.

```
1130 DIM RPS(2)
1140 M1 = 1
1150 M2 = 2
1160 M0 = 0
1170 & CNTFMT = M1,M2,M0
1180 FOR N = 0 TO 2
1190 & CLRCOUNTER
1200 & PAUSE = 1
1210 & COUNTERIN, (TV) = RPS(N)
1220 NEXT N
```

Line 1170 specifies the counter channels and the order in which they will be used by the sample loop in lines 1180 to 1220. For each pass through the loop, the & CLRCOUNTER command in line 1190 clears the counter and gets the "next channel" from the format statement. Input to that channel is then counted. The next reading (& COUNTERIN, line 1210) represents the count taken from that channel.

& COUNTERIN

&COUNTERIN, (TV) = avar

This command reads the accumulated count from the default (or otherwise specified) Counter Device. The count returned to (TV) will represent input to the counter channel specified in the most recently executed **& CLRCOUNTER** command line. If no channel number was included in the most recently executed **& CLRCOUNTER** command line, the count returned to (TV) will represent input to the "next channel" of the most recently executed **& CNTFMT** command. The automatic "cold start" execution of **& FMTDELT** will specify channel #7 as the counter channel to be read.

The default counter of the ISAAC model 91A is a 16-bit externally-clocked count-up device, so the raw value returned by **& COUNTERIN** will be an integer in the range of 0-65535 (-32767 to +32767 if Applesoft integer variables are specified).

The following parameters are optional with **& COUNTERIN**:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag (PR)		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

If **& COUNTERIN** occurs in a program after a cold start and before an **& CLRCOUNTER** command, the value returned will indicate the number of counts input to the designated counter channel since the time that LabSoft was initialized. This is likely to be pretty meaningless information. It's important to remember that there are two operations necessary to obtaining a valid count. The counter must first be cleared and connected to the desired input (using **& CLRCOUNTER**). Then, after a known interval, the count must be read (using **& COUNTERIN**). This is the only way to get accurate counts. You need to know what the counts represent, and you need to know when the counter was last cleared.

8.4

& FINL
& @FINL

& FINL, (TV) = avar
& @FINL, (AV) = numaryname

These commands read a frequency value in Hertz from the default (or otherwise specified) Counter Device. If a (C#) parameter is specified in the command line, that counter channel will be read. If no (C#) is specified, this command will use the "next channel" from the most recently executed & CNTFMT command. & FINL returns a raw value of 0-65535 Hertz. As always, you may use an optional function or other arithmetic operation to convert the raw value to any other units you'd like.

NOTE

These commands clear the counter as part of their operation. & CLRCOUNTER should not be used in conjunction with these commands.

The following parameters are optional with & FINL:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with & @FINL:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =
		Sampling RaTe	(RT) =
		No. of SWeeps	(SW) =

NOTE

You can also measure the frequency of very slow periodic waveforms using the & COUNTERIN command and a suitable delay loop. Such an arrangement will probably give more accurate counts for frequencies below 100 Hertz.

Software Frequency Measurement

LabSoft's frequency reading commands (& FINL, & @FINL, & FINH, and & @FINH) return a counts-per-period value. The length of that period determines whether counts-per-period translates into Hertz or kiloHertz. The internal operation of each iteration of this command involves the following sequence of events.

1. The counter is cleared.
2. Execution pauses for a fixed sample period (one second for & FINL, one millisecond for & FINH).
3. The accumulated count is returned as frequency.

In addition to providing an even more accurate reading of the revolutions-per-second of the high-speed motors in the program example used earlier in this chapter, & FINL would save time both in the writing of the program and its execution. Compare this program with the one listed under section 8.2. They both accomplish the same thing.

```
1130 DIM RPS(2)
1140 M1 = 1
1150 M2 = 2
1160 M0 = 0
1170 & CNTEMT = M1,M2,M0
1180 FOR N = 0 TO 2
1190 & FINL, (TV) = RPS (N)
1200 NEXT N
```

The counter input is not frequency range compensated in any way, nor does the command check for any minimum number of counts. No checks for inter-sample frequency stability are made. In other words, this is not a hardware frequency counter. It will return meaningful data only when uniform, glitch-free periodic waveforms are used as counter clock signals. Accuracy of the values returned is directly proportional to the frequency of the count. The routine is accurate to +/-1 count, therefore, the higher the frequency, the more accurate the value returned by this command.

8.5

& FINH
& @FINH

& FINH, (TV) = avar
& @FINH, (AV) = numaryname

These commands read a frequency value in kiloHertz from the default (or otherwise specified) Counter Device. If a (C#) parameter is specified in the command line, that counter channel will be read. If no (C#) is specified, this command will use the "next channel" from the most recently executed & CNTFMT command. & FINH returns a raw value of 0-65535 kiloHertz.

NOTE

These commands clear the counter as part of their operation. & CLR COUNTER should not be used in conjunction with these commands.

The following parameters are optional with & FINH:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with & @FINH:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	Channel #	(C#) =
		"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =
		Sampling RaTe	(RT) =
		No. of SWeeps	(SW) =

& FINH and & @FINH are the high-frequency-counting versions of & FINL and & @FINL. They sample for a period of one millisecond rather than one second. In all other respects, they operate exactly like their & FINL counterparts.

&&&

This Page Intentionally Left Blank



CHAPTER 9

THE TIMER AND CLOCK COMMANDS

- 9.1 & CLRTIMER
- 9.2 & TIMERIN
- 9.3 & TIME TO
- 9.4 & DAY TO

CHAPTER 9: THE TIMER AND CLOCK COMMANDS

In almost all data-acquisition applications, time is an important factor. All ISAAC systems have a timer/real-time clock device which can be used to time events, sample at designated times for designated intervals, record the day, date, hour, minute, and second, and handle just about any other aspect of timing/timekeeping. This device, located on the ISAAC/Apple Interface Board, contains a synchronous 16-bit count-up timer and an asynchronous battery backed-up real-time calendar-clock. Known by LabSoft as device number 15, this is the device that LabSoft's timer commands will access unless you specify otherwise with a (D#) parameter.

NOTE

The real-time clock commands will **only** access the default device. You may not specify any other device with either & TIME TO or & DAY TO.

The Timer Commands

9.1 & CLRTIMER

& CLRTIMER

& CLRTIMER clears the value of the default (or otherwise specified) timer/clock device's 16-bit count-up timer to zero (0). The only parameter optional with & CLRTIMER is (D#), which should be used if you need to have this command access a timer other than the one included in the default device.

9.2 & TIMERIN

& TIMERIN (TV) = avar

This command reads the current timer value from the default (or otherwise specified) timer/clock device. The default device has a timer with a resolution of 1.00352 milliseconds. & TIMERIN will return a raw value (in milliseconds) in the range 0-16383, which represents a real-time range of 0 to a little over 16 seconds.

NOTE

LabSoft uses the default timer to time matrix operations. Because of this, LabSoft clears the default timer at the start of all matrix I/O instructions. For this reason, you cannot use the timer commands & CLRTIMER and & TIMERIN to critically time sections of BASIC code that contain LabSoft matrix I/O commands. If such timing is necessary, it will have to be accomplished using the real-time clock commands, which have a maximum resolution of 1 second.

The following parameters are optional with & TIMERIN:

Analog Graph Flag	(GA)	Device #	(D#) =
Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
		"XOR" Mask	(XM) =
		Compare Value	(CV) =

The Real-Time Clock Commands

9.3 & TIME TO

& TIME TO avar1, avar2, avar3

& TIME TO returns the current time of day, as measured by the ISAAC 91A's default real-time calendar/clock, to the three specified variables. It returns the hours to avar1, the minutes to avar2, and the seconds to avar3. Three variables must always be specified with this command. Specifying more (or fewer) than three variables will generate a ?SYNTAX ERROR. No parameters are optional with & TIME TO.

Like any good clock, ISAAC's real-time clock must first be set to the correct time. In addition (and unlike all but the very finest clocks), either a 12 hour or 24 hour format may be selected. The LabSoft Master Disk includes a program (SETCLOCK) for this purpose.

9.4 & DAY TO

& DAY TO avar1, avar2, avar3, avar4

& DAY TO returns the current date, as measured by the default real-time calendar clock, to the four specified variables. It returns the year to avar1, the month to avar2, the day of the month to avar3, and the day of the week (expressed as a numeric value where Sunday = 0 and Saturday = 6) to avar4. Four variables must always be specified with this command. Specifying more or less than four variables will generate a ?SYNTAX ERROR. No parameters are optional with & DAY TO.

The real-time clock commands are particularly useful for time/date-stamping data as it is acquired. These commands can also be used to initiate operations at certain predetermined hours, sound beeper/buzzer alarms, or time events of longer duration where resolution greater than one second is not necessary.

&&&

This Page Intentionally Left Blank



CHAPTER 10

& LOOK FOR...THEN...

- 10.1 Looking For an Input
- 10.2 Looking For an Output

CHAPTER 10: & LOOK FOR... THEN...

One of the most useful features of LabSoft is the ability to perform triggered operations. By triggered operations, we mean operations that are executed only when some specified condition is met.

Anyone who has spent hours in a laboratory waiting for some critical moment in a process or experiment knows that a triggered input can be a valuable thing. And anyone who has ever had to initiate some event and almost simultaneously begin to record data generated by that event has a good idea of the potential uses of output triggering. Once you learn how to use LabSoft's triggered (& LOOK FOR... THEN...) command structure, you'll probably come up with a variety of ways to use it to make your data acquisition and control chores much easier.

& LOOK FOR... THEN... expressions permit any detectable event to "trigger" any LabSoft or Applesoft command. Any of LabSoft's I/O commands can be used, along with a threshold value (TH) parameter, as the trigger (the "looked for") part of the expression. When the specified threshold value is met or exceeded, any valid LabSoft expression which follows will be executed in a few microseconds. If no (TH) parameter is specified, any input of the type specified will cause the THEN... portion of the command to be executed. While not "required" in the strictest sense, (TH) is an extremely valuable parameter in this type of expression.

The required syntax for & LOOK FOR... THEN... expressions is:

& LOOK FOR any LabSoft I/O command expression
[, (TH) = aexpr] : any valid Applesoft or LabSoft expression.

This is an example of a "balanced" command expression. The colon separates the two halves of the expression. We will refer to these two

halves as the "trigger" (or "LOOK FOR") half and the "triggered" (or "THEN") half.

Here's an example which may make this command's syntactic requirements clearer. The following program line will order the system to wait until the analog input value exceeds 2000, then sound ISAAC's buzzer.

& LOOK FOR AIN, (TV)=Q, (C#)=0, (TH)=2000 : & BUZZ

NOTE

There is no command word THEN in the expression. The THEN is implied by the colon (:), Apple-soft's "end of statement" character.

**Caution

The ampersand (&) preceding LOOK FOR eliminates the need for any other ampersand in the TRIGGER part of the expression. If the second half of the expression (following the :) is a LabSoft command, use of the ampersand will again be necessary.

& LOOK FOR... THEN... behaves somewhat differently when it is LOOKing FOR an input than when it is LOOKing FOR an output. We'll examine each of these uses in turn, and give our usual couple of examples.

10.1

LOOKing FOR an Input

When a LabSoft input command is used as the triggering half of a & LOOK FOR... THEN... expression, we refer to this arrangement (logically enough) as input triggering, or LOOKing FOR an input. Any of the following LabSoft input commands may be used as the trigger half of & LOOK FOR... THEN expressions:

& AIN	& ASUM
& BIN	& BPOLL
& BCDIN	& COUNTERIN
& FINL	& FINH
& TIMERIN	& TRIGIN
& TRIGPOLL	& RDEV

For an example of input triggering, assume that you need to look for an analog value (input to Channnel #3 of Device #3) that is greater than half of its full scale and, when that value is

reached, print the message "GOT IT". Here's all the code you'll have to write:

```
770 & LOOK FOR AIN,(D#) = 3, (C#) = 3, (TV) =  
X, (TH) =2049 : PRINT "GOT IT"
```

When line 770 is executed, the following sequence of events takes place:

1. All LabSoft commands in **both halves** of the command line are interpreted. Any Applesoft commands (PRINT, in this instance) in the THEN half are not interpreted yet.
2. A raw analog value (in this case 0-4095 from a 12-bit A/D) is continuously input and compared to the specified threshold quantity (2049 in our example).
3. When the masked raw input value is found to be greater than 2049, the second half of the command line will be executed. In the example above, this command is an Applesoft command, PRINT, which must first be interpreted, then executed. LabSoft commands in the "THEN" half of these expressions are pre-interpreted and execute at substantially faster speeds.

The "LOOK FOR" section of the command compares the integer value specified by the threshold (TH) parameter with the raw (or masked) integer value input. This integer comparison might seem somewhat inconvenient, since it requires that you work with raw masked values instead of real values converted by an optional function. However, this "inconvenience" actually facilitates operations. Internally LabSoft can perform an integer comparison in a matter of microseconds; comparisons involving reals take substantially longer. The & LOOK FOR... THEN... structure's primary advantage is speed, and its operation has been designed so that the time between the receipt of a valid trigger and the "THEN" operation is as short as possible; a matter of microseconds. It would take many milliseconds to emulate the operation of this type of expression using the (CV) parameter and BASIC evaluation.

The simple comparison for "greater than" or "equal to" may seem like a limitation at first, but it really isn't when you consider that the LOOK FOR operation is comparing the raw masked input value with the integer threshold value. You could "XOR" the input to invert it, thereby effectively LOOKING FOR a "less than" condition. You could even selectively mask out certain bits and LOOK FOR equality. Any extra time spent in setting up this routine will be amply rewarded by increased speed of operation.

NOTE

The comparison done during execution of the "LOOK FOR" statement should not be confused with the comparison done by the optional parameter (CV). & LOOK FOR... THEN does not in itself affect the reserve variable flags LT%, EQ% or GT%.

Here's another example. Suppose you need to matrix input a number of analog values, but only after a signal from an external source is received by an ISAAC binary input device. The next program allows you do this, using only two lines of code.

```
2060 DIM A (999)
2070 & LOOK FOR BIN, (TV) = X, (TH) = 1:
      & @AIN, (AV) = A, (RT) = 10
```

When a & LOOK FOR... THEN... expression is encountered in a program, the first thing that happens is that all LabSoft commands in the expression are interpreted. After that, the actual execution begins. This way, execution of the triggered half of the expression will follow a valid trigger without any delay due to interpreter overhead (the time it takes for a command to be interpreted). If the triggered statement is an Applesoft expression (rather than a LabSoft one) it will not be interpreted until after LabSoft has found whatever it was told to LOOK FOR. In this case, execution of the triggered portion of the command will be somewhat delayed while the Applesoft statement is interpreted.

NOTE

If the triggering expression uses a matrix input command, the input and comparison operations will take place for each element of the target array. Each time the trigger condition is met,

the current value will be stored in the next element of the array. This process will continue for all elements of the array. Only after the last element in the array has been filled will the triggered expression be executed. In most cases, you'll want to avoid using matrix commands as triggers in & LOOK FOR... THEN... expressions. If you want to sample an input repeatedly and only save those values that meet threshold conditions, you could use a matrix command as the trigger and a "dummy" BASIC statement in the THEN expression (a REM would do).

10.2 LOOKING FOR an Output

Another way to use & LOOK FOR... THEN... expressions is to trigger the "THEN" half of the expression with an **output** command. This may at first seem a little pointless, since in essence, LabSoft would be looking for a value that it would (at some point) have to generate internally. LabSoft would be looking for something it already had.

But the execution speed inherent in this structure makes it an ideal format for triggering an external event with an output value (& LOOK FOR...), then (THEN..) almost simultaneously starting to acquire data generated by that triggered event.

The following output commands may be used as triggers in & LOOK FOR... THEN expressions:

& AOUT	& BOUT
& BCDOUT	& WRDEV

A typical application for output triggering might use a line of code like line 900 below to send a 5 Volt analog signal through D/A channel #1 of Device #0, then begin polling the binary inputs, looking for one to go high.

```
900 & LOOK FOR AOUT, (DV) = 4095, (D#) = 0,  
      (C#) = 1 : & BPOLL, (TV) = X
```

Using & LOOK FOR... THEN... in this way allows you to use ISAAC to both initiate events in the

and to report on their consequences. Best of all, it lets you do these two things in extremely rapid succession.

NOTE

You need not specify a threshold value (TH) for an **output** command used as a trigger. If you do, it will be ignored.

When used with output command triggers, a **& LOOK FOR... THEN...** expression will be executed in the following sequence.

1. The "LOOK FOR" half of the expression is interpreted.
2. The "THEN" half of the expression is interpreted (only if it is a LabSoft command).
3. Both halves of the expression are executed: the "LOOK FOR" (trigger) half first, followed (a few microseconds later) by the "THEN" (triggered) half. If the triggered command is an Applesoft command, it will not be interpreted until after the trigger command is executed. This will slow overall execution time somewhat.

A Final Word

& that's about all there is to LabSoft. Of course, there's more information in the appendices to this manual, and still more in the User's Guide and System Reference (Hardware Reference), so please have a look at them. And don't forget that a System Index, covering all three volumes of ISAAC documentation, is included in the User's Guide. It's particularly useful in that it locates **examples of use** of various commands that are scattered throughout the ISAAC reference library.

As is true of all reference manuals, this one should be kept handy, so that you can refer to it whenever you have a question about exactly how a certain command works. Nobody is expected to memorize the whole thing. We have to look things up in here ourselves from time to time. You probably will too.

&&&



APPENDICES

APPENDIX A: ERROR MESSAGES GENERATED BY LABSOFT

Since LabSoft is an extension of Applesoft BASIC, all of the possible error conditions generated by Applesoft should already be familiar to the LabSoft programmer. A complete list of Applesoft error messages can be found in the Applesoft Reference Manual. LabSoft operations tend to generate the following kinds of errors:

Error Number	Error Type
16	SYNTAX ERROR
42	OUT OF DATA ERROR
53	ILLEGAL QUANTITY ERROR
77	OUT OF MEMORY ERROR
107	BAD SUBSCRIPT ERROR
163	TYPE MISMATCH ERROR
176	STRING TOO LONG ERROR
191	FORMULA TOO COMPLEX ERROR
224	UNDEFINED FUNCTION ERROR

&&&

APPENDIX B: LABSOFT RESERVED VARIABLES

LabSoft makes use of a number of reserved variables which are reserved (as their name implies) for the storage of LabSoft values. Your program can read values from any of the reserved variables and use them in a number of ways (formulas, etc.). Since LabSoft stores all kinds of I/O data in these variables, it is ordinarily not a good idea to use them to store anything you'd mind losing.

The following table lists each reserved variable, the type of data it normally contains, and the LabSoft instructions which affect it.

Variable	Contents	Commands
RA(W)%	Raw value input from hardware by any of the input I/O comands. Also, binary value to be output by output I/O commands.	& BIN, & BOUT & ASUM & TIMERIN & COUNTERIN & TRIGIN & TRIGPOLL & BPOLL & RDEV, & WRDEV & AIN, & AOUT & BCDIN & BCDOUT
MA(SKED)%	Result of raw value masked with the optional "AND" and "XOR" masks in all I/O commands.	(same as RAW% above)
OU(T)	Value of data to be output. (May be the argument of an optional function)	& BOUT & AOUT & WRDEV & BCDOUT

Variable	Contents	Commands
LT%	If an optional compare value (CV) is specified in input I/O commands, these three reserved variables will indicate the result of the comparison. Their names stand for: Less Than, Equal to, and Greater Than. If the value input is represented by IN, and if the compare value is represented by CV, an input I/O command will set these values in the following states:	& BIN, & AIN
EQ%		& TIMERIN
GT%		& COUNTERIN
		& ASUM
		& TRIGPOLL
		& BPOLL
		& RDEV, & BCDIN

IF	THEN
IN < CV	LT% = 1 EQ% = 0 GT% = 0
IN = CV	LT% = 0 EQ% = 0 GT% = 0
IN > CV	LT% = 0 EQ% = 0 GT% = 1

NOTE When using the matrix commands, you may occasionally notice a delay between the initial interpretation of a command and its execution. You may also notice that this occurs only when your command is used with a very large array and when LabSoft's reserved variables (RAW%, MASKED%, OUT, LT%, EQ%, and GT%) have not yet appeared in the program. This happens because of the way simple variables and arrays are stored in memory by BASIC.

You can eliminate these "side effects" by performing one of the following commands early in your program, before any arrays are dimensioned:

- A. "Declare" all of the reserved variables. Although BASIC doesn't have declaration statements, a line of the form:

```
100 RAW% = 0 : MASKED% = 0 : OUT = 0 : LT% = 0
    :EQ% = 0 : GT% = 0
```

will suffice.

- B. Execute any I/O command. A simple input command like:

```
100 & AIN,(TV) = X,(C#) = 1
```

will do.

Either of these must be done BEFORE any arrays are dimensioned. Both will inform BASIC of reserved variable placement and will do away with execution lag time.

&&&

APPENDIX C: LABSOFT'S OBJECT FILES

The LabSoft Master Disk contains two different versions of LabSoft. The first, called LABSOFT.ROM.OBJ, loads into program memory and lives just under DOS with three buffers. The second, LABSOFT.RAM.OBJ, loads into a language card if one is present. The LABSOFT HELLO program on the LabSoft Master Disk is designed to check the configuration of any system on which it is booted, then load and initialize the appropriate LabSoft object file.

If for some reason, you are running a program that requires the value of MAXFILES (i.e., the number of DOS buffers) to be greater than three, you will have problems if LABSOFT.ROM.OBJ is resident in program memory.

If your system has a language card, these problems (and a few others) will never bother you. So if it's at all possible, you should acquire one and install it in Apple Peripheral Slot #0

&&&

APPENDIX D: MEMORY MAPS

With LABSOFT.ROM.OBJ loaded and initialized, the Apple's memory will look like this.

\$ FFFF	MONITOR ROM
\$ F800	APPLESOFT BASIC
\$ D000	EXPANSION SLOTS
\$ C800	APPLE I/O HARDWARE ADDRESSES
\$ C000	DOS
\$ 9600	3 DOS BUFFERS
\$ 7000	LABSOFT
	BASIC STRINGS
	BASIC VARIABLES
\$ 800	BASIC PROGRAM
\$ 400	SCREEN MEMORY
\$ 380	LABSOFT VECTORS
\$ 300	APPLE PATCH AREA
\$ 0	APPLE SYSTEM AREA

NOTE LabSoft reserves the upper part of page 3 (\$3C0 - \$3CF) for its vectors. Since there are some utilities and patches that use this page, you must be careful, when using them, not to write over this area. If you think you may have done so, re-boot LabSoft to be sure these vectors are present. LabSoft won't work properly without them.

With LABSOFT.RAM.OBJ loaded and initialized, the Apple's memory map will look the figure on the next page.

\$ FFFF	MONITOR ROM	
\$ F800	APPLESOFT BASIC	LABSOFT
\$ D000	EXPANSION SLOTS	
\$ C800	APPLE I/O HARDWARE ADDRESSES	
\$ C000	DOS AND DOS BUFFERS	
	BASIC STRINGS	
	BASIC VARIABLES	
\$ 800	BASIC PROGRAM	
\$ 400	SCREEN MEMORY	
\$ 380	LABSOFT RESERVED AREA	
\$ 300	APPLE PATCH AREA	
\$ 0	APPLE SYSTEM AREA	

This version of LabSoft uses half of page 3 (\$380 - \$3FF) for itself. Again, be careful not to use this area for any other patches or utilities. Note also that loading and initializing this version of LabSoft will "step on" all of page 3.

&&&

APPENDIX E: SOME USEFUL PEEKS, POKES AND CALLS

In the Applesoft Reference Manual, there is an appendix on PEEKS, POKES and CALLS. Each of these performs some function useful to the programmer, but due to space limitations or other considerations, they are not listed as full-fledged commands.

LabSoft has its own PEEKs, POKEs, and CALLs. Like Applesoft's, they will be particularly useful to the experienced programmer. Some of the more helpful ones are listed here.

E.1 CALL 972

CALL 972 Initializes LabSoft. This initialization performs all of the following operations.

1. Sets the Applesoft Ampersand jump vector to point at LabSoft.
2. Writes LabSoft's important jump vectors and addresses into the top of page 3, just below the DOS vectors.
3. Resets all of LabSoft's formats and internal variables to their default values (the same as executing an &FMTDFLT command).
4. Clears all CLEAR TO SEND lines to their "not ready" state.
5. Sets all LabSoft internal constants and variables to their default state.
6. In the Language Card version of LabSoft, CALL 972 writes both the bank switch vectoring routines and the Version and Release header into page 3. In the program memory version, CALL 972 resets HIMEM to just below LabSoft's memory location.

E.2 `PLT = PEEK (966) + PEEK (967) * 256`
 `POKE PLT, 0`

This procedure sets the LabSoft Graphing routine to a "point plot" mode. Normally, LabSoft's graphing routine will draw a line between plotted points, producing a "continuous" graph. This kind of graph looks good, but unfortunately, drawing it this way occupies a great deal of CPU time. Changing the display to a "point plot" mode will speed things up a bit.

NOTE In the Language Card version of LabSoft, this location cannot be PEEKed at from BASIC.

E.3 `PLT = PEEK (966) + PEEK (967) * 256`
 `POKE PLT,1`

This POKE resets LabSoft's graphing routine to the default "line plot" mode.

E.4 `PLT = PEEK (966) + PEEK (967) * 256`
 `POKE PLT,2`

This POKE changes the graphing routine so that a histogram-type display is produced. In this mode, a line is drawn from each point to the bottom of the graph. This mode is "slower" (i.e., it occupies more CPU time) than any other graphing routine, but it provides a more visually striking display when one curve is being plotted. When more than one curve is being plotted, however, this POKE will result in a very confusing display.

E.5 `DIV = PEEK(966) + PEEK(967) *256 + 3`
 `POKE DIV, n`
 `POKE DIV +1, n`
 `POKE DIV +2, n`
 `POKE DIV +3, n`
 `POKE DIV +4, n`

This series of POKES can be used to change the divide-down values used by the (GA) flag. If you set this optional flag in one of the I/O commands, LabSoft will automatically plot the value on a graph. The LabSoft graphs only have a vertical range of 128 points or 7 bits. In order to plot values that have a potential range greater than 7 bits, LabSoft divides the values down. For example, the analog inputs and outputs are 12-bit

numbers. LabSoft divides these numbers down by 5 bits, so that the maximum A/D count of 4095 divides down to a graph value of 127. This process permits the entire range of the A/D to be displayed on the graph.

Now and then, you may want to expand the graph for increased resolution. You can do this by decreasing the divide-down value. For every unit that the divide-down value is decreased, the graph is expanded by a factor of two. In the example above, if you changed the analog divide-down from 5 to 4, the graph would only cover half of the A/D range, but it would do so with twice the resolution. This condition would be fine if you were only measuring values between -5 and 0 Volts in a normally configured system.

These five POKes can be used as shown here to change the divide-down values.

Type of Operation	Default Value	POKE
BCD	7	POKE DIV, n
DEVICE	9	POKE DIV +1, n
TIMER	9	POKE DIV +2, n
BINARY	9	POKE DIV +3, n
ANALOG	5	POKE DIV +4, n

NOTE In the Language Card version of LabSoft, these locations cannot be PEEKed from BASIC.

E.6 $EN = \text{PEEK}(970) + \text{PEEK}(971) * 256$

A statement of this form will return the address of the byte following the last byte of the LabSoft object file currently in the system. This is also the address of the first byte of the & LABEL command's character set, and the address at which any replacement character sets should be located.

E.7 $VER = \text{PEEK}(968) + \text{PEEK}(969) * 256$

A statement of this form will return (to the variable VER) the address of the first byte of the Version and Release header of the LabSoft object file currently in memory. This header is a string which contains information about the file's version number and release date.

A typical header will look like this.

V 1.0 R 01/01/81

&&&

APPENDIX F: SPEEDING UP LABSOFT PROGRAMS

The Applesoft Reference Manual offers some very useful hints on optimizing your BASIC programs for speed. These tips also apply to LabSoft programs, and there are some additional measures you can take to speed execution of LabSoft routines.

Do as Little as Possible in Each Command

As you can determine by looking at the benchmarks in Appendix H of this manual, it is time-consuming to do a sum, evaluate an optional function or to do a value comparison. It takes even more time to update a graph display, particularly if the graph is the scrolling type.

When designing and writing LabSoft I/O routines in which speed is an important factor, try to postpone all non-essential operations until after you have collected data. For example, don't evaluate each data observation in real-time unless absolutely necessary. Remember, as the benchmarks in Appendix H show, simple inputs to and from integer variables are easily the fastest LabSoft operations available.

Use Matrix (Array) Operations Instead of Simple Commands in FOR...NEXT Loops

Applesoft FOR...NEXT loops are slow. Each command must be interpreted and executed on each pass through the loop. Operations will be faster if you substitute Matrix I/O commands whenever and wherever possible.

Use Integer Variables Whenever Possible

Unlike most Applesoft operations, where the use of integer variables actually slows a program somewhat, most LabSoft operations will execute faster when integers are used in place of real values. You will notice this especially when using matrix ASUM operations. As noted in Appendix G, this technique could save you a considerable amount of

space as well as time. Be aware, however, that you may run into a problem with equivalent integer values. This is more fully explained in Appendix M.

NOTE Most operations will run a little bit faster if LABSOFT.ROM.OBJ is loaded and initialized. This is because there is some extra software overhead involved in switching back and forth between the language card and the Apple's main memory. This difference in speed is hardly worth mentioning. At its most noticeable (during graphing or arithmetic operations) it is no more than a minor annoyance. But we thought you ought to know.

&&&

APPENDIX G: MAKING EFFICIENT USE OF MEMORY

Keeping the Programs Compact

Appendix D of the Applesoft Reference Manual provides some valuable tips for making BASIC programs more compact; tips which apply to LabSoft programs as well. In addition, we would like to offer two additional suggestions that may help you to save memory space when writing and running LabSoft programs.

1. Save large raw data arrays **only** when you really need the data in raw form. In many applications, large data arrays are collected principally for the purpose of deriving an average. If an average of many data values is all you need it might be best to use LabSoft's &ASUM command. This command, which stores a sum of data as a two-byte word, will be particularly advantageous at lower sampling speeds.
2. Avoid unnecessary graphing. LabSoft graphing commands (like Applesoft HIRES graphics routines) require 8K of memory for **each** high-resolution screen. All or part of that space can be made available for your program and variables if you either avoid using graphics altogether, or follow one of the memory-management schemes below.

Memory Considerations When Using Graphics

Unfortunately, Apple II hardware maps high resolution graphics screen #1 from 8K to 16K, and maps screen #2 from 16K to 24K in memory. This is right in the middle of the area in memory where BASIC programs and variables normally go. So, when high resolution graphics (this includes **any** LabSoft graphing routine) are included in a program, the size of the program and the extent of its' variable tables will be limited. To keep BASIC text, variables and the high resolution screens out of each other's way, you may have to resort to some form of **memory**

management. Details on the instructions used here (PEEK, LOMEM, CLEAR) are included in the Applesoft Reference Manual.

The principal things to remember about memory structure in the Apple are:

- the BASIC program begins at memory location 2049 (\$801 hex) and build up,
- the variable table begins at LOMEM, which (unless otherwise specified by the LOMEM command) is automatically set to the next byte after the last byte of the BASIC program.
- BASIC strings begin at HIMEM, which (unless otherwise specified by the HIMEM command) is automatically set to the highest available RAM location, and build down.

Interference between a program or data and either of the high-resolution graphics screens can have a number of consequences, all of them undesirable. Users of long programs that collect a lot of data may need to resort to a memory management scheme.

You can use Applesoft's LOMEM command to state explicitly the location where data storage will begin. Doing this will protect one (or both) of the high-resolution screens from being overwritten by **data**.

Before doing this, though, you should check the End Of Program (EOP) pointer, using the BASIC statement:

```
] PRINT PEEK (175) + PEEK (176) * 256
```

This statement returns the highest address of the program currently in memory, a value we'll refer to as EOP.

If EOP < 8192

This means that the program doesn't overwrite either of the high-resolution screens. Data still might, however, depending on the length of

the program and the amount of the data. If EOP is, say, <1000 and you expect only a small amount of data to be written, you will have no problem with data overwriting the graphics screen. To a certain extent, you'll have to use your own judgement here.

If you suspect that your data will overwrite high-resolution graphics screen #1, and you need both high-resolution graphics screens for your program, set LOMEM above screen #2 using the following two lines.

```
10 LOMEM: 24579
20 CLEAR
```

If you only need high-resolution graphics screen #1 for your program, you can gain an extra 8K of memory for your variable table by beginning the program with the following lines.

```
10 LOMEM: 16384
20 CLEAR
```

If EOP > 8192 AND EOP < 16384

This means that your program has overwritten high-resolution graphics screen #1. If this is the case, you should begin the program with the following lines.

```
10 LOMEM: 24579
20 CLEAR
```

This will keep high-resolution graphics screen #2 from being overwritten by data. But since high-resolution graphics screen #1 has been overwritten by the program, you'll have to do all your graphing on screen #2.

If EOP > 16384

If this is the case, you will need to use the APPLE SLICER memory-management program on the LabSoft Utility Disk.

APPENDIX H: LABSOFT BENCHMARKS

The following benchmarks represent single cycle sampling rates for LabSoft matrix commands operating with a free-running sampling rate, i.e., (RT) = 0. Not all I/O parameter permutations were benchmarked, but you should be able to get some idea of how much time is required for each command type and each parameter option from this data.

Please be aware that you should not depend on free-running sampling rates as they may change with each new version of LabSoft. In addition, you should realize that the free-running execution time of some of the commands is not consistent. In these cases, the benchmark given is an appropriate average.

These figures are offered only to give you an idea of the maximum rates that may be used in various command configurations. Do not depend on them. Also note that there are some optional parameters (the masks) which do not affect sampling rate at all.

COMMAND	AVERAGE SINGLE EXECUTION TIME IN MILLISECONDS
& @BIN,(AV) = X%	.56
& @BOUT,(AV) = X%	.49
& @TRIGIN,(AV) = X%	.58
& @BCDIN,(AV) = X%	5.6
& @BCDOUT,(AV) = X%	4.0
& @AIN,(AV) = X%	.84
& @AOUT,(AV) = X%	.54
& @AIN,(AV) = X	1.6
& @ASUM,(AV) = X	1.9
& @AIN,(AV) = X, (CV) = 2048	1.9
& @AIN,(AV) = X,(FU) = FN X(MASKED%)	32.2
& @AIN,(AV) = X,(GA) ALTERNATING GRAPH	3.2
& @AIN,(AV) = X,(GA) SCROLLING GRAPH	150.0
& @AIN,(AV) = X,(PR) PRINT TO DISK FILE	29.0
& @BIN,(AV) = X,(GB)	15.0
& LOOK FOR BOUT,(DV)=1 : & BOUT,(DV)=0	1.3

NOTES 1. The function used in the analog input with optional function benchmark was a standard scaling function, containing a scale and offset

constant. Since functions are literally interpreted in real-time, the length of time required for the evaluation of a function is extremely variable. In general, the more complex the function, the longer the evaluation time.

2. The execution time given for the analog input with scrolling graph benchmark applies to graphs that are actually scrolling. Full-screen graphs that do not scroll take much less time. Their execution occurs more in the range of the analog input with alternating graph benchmark.
3. The analog input with the print to disk file benchmarks may be of use to those who wish to acquire data and dump it directly to a disk file. The benchmark represents the average execution time for one input sample and writing the raw value to disk as ASCII, with a carriage return character. These speeds are for Apple Disk II with 5-inch floppies.
4. These benchmarks were calculated using the language card version of LabSoft. In general, execution speeds will be faster in the program memory version, particularly where graphing or arithmetic operations are concerned.
5. The execution time for printing to disk file will vary significantly, depending on the number of digits in each value. Disk operations in general are very irregularly timed. Each value is first stored in a buffer (a relatively quick operation) which, when it is filled, is written to the disk (a relatively slow operation). Our benchmarks represent an average of these operations for a large number of points.
6. The LOOK FOR...THEN... benchmark represents the time between execution of the first & BOUT command and the second. This time will vary, depending on the commands used. It is safe to assume that an & LOOK FOR...THEN... construction will execute at least ten times faster than the simplest possible alternative construction.

&&&

APPENDIX I: GENERATING YOUR OWN CHARACTER SET
(For Use With LabSoft's & LABEL Command)

The character set provided on the LabSoft Master Disk for use with & LABEL was generated by the utility program ANIMATRIX. This utility is one of a collection of programs available from Apple Computer Company. The collection is called the Apple-soft Tool Kit, and you can get it at your computer store.

With ANIMATRIX you could easily generate character sets for use with & LABEL. However, you cannot use them as they are generated by ANIMATRIX, because LabSoft requires its character sets to have a header. The following instructions explain how to put this header on any character set generated by ANIMATRIX.

NOTE These modified character sets cannot be used with any language other than LabSoft.

1. Generate and save on disk a character set using the ANIMATRIX program.
2. Get your machine into the immediate BASIC mode.
3. Set HIMEM to 32768 with an immediate BASIC statement:

```
]HIMEM: 32768
```

4. BLOAD your text set file at 32773 with an immediate DOS command:

```
]BLOAD LABEL. SET,A 32773
```

5. Poke the header by entering the following immediate BASIC statements:

```
]POKE 32768,76 : POKE 32769,65  
]POKE 32770,66 : POKE 32771,69  
]POKE 32772,76
```

6. BSAVE the text set with header with the immediate DOS command:

]BSAVE LABEL. SET,A32768,L765

7. Restore HIMEM by executing the immediate DOS command:

]FP

8. Your text set is now ready to use with Lab-Soft's LABEL command.

&&&

APPENDIX J: BIT MASKING

Here are a couple of binary-decimal conversion tables you may find useful when you're using (AM) or (XM) parameters.

(AM) = Highest-Order Unmasked Bit

1	0
3	1
7	2
15	3
31	4
63	5
127	6
255	7
511	8
1023	9
2047	10
4095	11
8191	12
16383	13
32767	14
65535	15

Individual Bit Values (when high)
Bit # Bit value

0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32767

(AM) =	Lowest-Order Unmasked Bit
32768	15
49152	14
57344	13
61440	12
63488	11
64512	10
65024	9
65280	8
65408	7
65472	6
65504	5
65520	4
65528	3
65532	2
65534	1
65535	0

&&&

APPENDIX K: A LABSOFT DICTIONARY

An Alphabetical List of LabSoft Parameters

Parameter Label	Parameter Range	Parameter Definition
(AM) =	integer 0 - 65535	logical "And" Mask
(AV) =	numeric array name	Array Variable
(CV) =	any numeric value	Compare Value
(C#) =	integer 0 - 255	Channel #
(DV) =	any numeric value *	Data Value
(D#) =	integer 0 - 15	Device #
(FU) =	any legal function name	FUnction
(GA)	(none required)	Graph Analog
(GB)	(none required)	Graph Binary
(PR)	(none required)	Print Raw
(RT) =	integer 0 - 16000	sampling RaTe
(SW) =	integer 1 - 65535	number of SWeeps
(TH) =	0 - 65535	THreshold value
(TV) =	integer or real variable	Target Variable
(W#) =	integer 0 - 15	Word #
(XM) =	integer 0 - 65535	logical "Xor" Mask

All parameter labels must be preceded by a comma.

Although parameters (GA), (GB), and (PR) may all be specified on a command line, only the LAST ONE SPECIFIED will be executed.

* final output value cannot exceed 65535

Syntax Definitions

[expressions within these brackets are optional expressions]

/ is defined as the logical "OR"

numaryname is defined as the name of a numeric array

aexpr is defined as an arithmetic expression

sexpr is defined as a string expression

An Alphabetical List of LabSoft Commands

& AIN,(TV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(C#)]

Inputs an analog value into (TV).

& @AIN,(AV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(C#),(RT),(SW)]

Matrix version of & AIN.

& ALTSET

Initializes Alternating graph.

& ANAFMT = aexpr1 [,aexpr2...,aexprN]

Specifies the analog channel format.

& AOUT,(DV) [(C#),(D#),(FU),(AM),(XM),(GA)/(GB)/(PR)]

Analog output of value (DV).

& @AOUT,(AV) [(C#),(D#),(FU),(AM),(XM),(GA)/(GB),(PR),(RT),(SW)]

Matrix version of & AOUT.

& ARRAYCLR, numaryname

Resets all elements of numaryname to 0.

& ASUM,(TV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(C#),(RT),(SW)]

Inputs analog value and sums it with current value of (TV).

& @ASUM,(AV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(C#),(RT),(SW)]

Matrix version of & ASUM.

& BCDIN,(TV) [,D#),(FU),(AM),(XM),(GA)/(PR),(CV)]

Inputs a 16-bit binary word as an unsigned 4-digit BCD value.

& @BCDIN,(AV) [, (D#),(FU),(AM),(XM),(GA)/(PR),(RT),(SW)]

Matrix version of & BCDIN.

& BCDOUT,(DV) [, (D#),(FU),(AM),(XM),(GA)/(PR)]

Outputs an unsigned 4-digit number as a 16-bit BCD value.

& @BCDOUT,(AV) [, (D#),(FU),(AM),(XM),(GA)/(PR)]

Matrix version of & BCDOUT.

& BEEP

Beeps ISAAC's internal speaker .1 second.

& BEEP ON

Beeps ISAAC's internal speaker continuously.

& BEEP STOP

Cancels an & BEEP ON command.

& BIN,(TV) [, (D#),(FU),(AM),(XM),(GA)/(GB)/PR,(CV)]

Inputs a 16-bit binary value into (TV).

& @BIN,(AV) [, (D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(RT),(SW),(CV)]

Matrix version of & BIN.

& BINFMT = aexpr1 [,aexpr2,...aexprN]

Specifies the binary graphing format.

& BOUT,(DV) [,(D#),(FU),(AM),(XM),(GA)/(GB)/(PR)]

Outputs (DV) as a 16-bit binary word.

& @BOUT,(AV) [,(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(RT),(SW)]

Matrix version of & BOUT.

& BPOLL,(TV) [,(D#),(FU),(AM),(XM),(PR),(CV)]

Waits for a binary input line to go high,
then returns to (TV) the bit number of the one
that did.

& @BPOLL,(AV) [,(D#),(FU),(AM),(XM),(PR),(CV),(RT),(SW)]

Matrix version of & BPOLL.

& BUZZ

Buzzes ISAAC's internal speaker .1 second.

& BUZZ ON

Buzzes ISAAC's internal speaker continuously.

& BUZZ STOP

Cancels an & BUZZ ON command.

& CLRCOUNTER [,(D#) ,(C#)]

Clears the counter to 0 and sets the counter channel
to be read.

& CLRTIMER [(D#)]

Clears the timer to 0.

& CNTFMT = aexpr1 [,aexpr2...aexprN]

Specifies the counter channels format.

& COUNTERIN,(TV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV)]

Reads the counter and clears it to 0.

& DAY TO avar1,avar2,avar3,avar4

Reads the year, month, day of the month, and day of the week from the real time clock.

& ERRPTCH

Executes the ONERR... GOTO...error-patching routine described in the Applesoft Reference Manual.

& FINH,(TV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(C#)]

Returns a value (in KiloHertz) for the frequency of the signal at the counter input.

& @FINH,(AV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(C#),(RT),(SW)]

Matrix version of & FINH

& FINL,(TV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(C#)]

Returns a value (in Hertz) for the frequency of the signal at the counter input.

& @FINL,(AV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(C#),(RT),(SW)]

Matrix version of & FINL

& FMTDFLT

Resets all formats as shown:

& CNTFMT=7

& BINFMT=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

& PLTFMT=1

& ANAFMT=0

& FULLSCREEN

Selects full screen graphics.

& HIRES1

Selects the Apple's HIRES screen #1 without erasing it first.

& HIRES2

Selects the Apple's HIRES screen #2 without erasing it first.

& LABEL = sexpr AT aexpr1,aexpr2

Places a string, starting at the specified location on the current HIRES display.

& LOOK FOR LabSoft I/O command without '&' [(TH)] :
any valid Applesoft/LabSoft expression

Triggers the second expression when the value of (TH) is met or exceeded. Default value of (TH) = 0.

& MIXSCREEN

Selects mixed screen graphics; allows four lines of text at the bottom of the screen.

& NXTBIN = aexpr

Plots the next point on the currently defined binary graph.

& NXTPLT = aexpr

Plots the next point on the currently defined analog graph.

& OFFERR

Cancels ONERR...GOTO... status

& OUTLINE

Outlines the currently initialized graph in the currently defined HCOLOR.

& PAUSE = aexpr

Pauses program execution for the number of seconds and tenths of seconds specified by aexpr.

& PLTFMT = aexpr1 [,aexpr2...aexpr16]

Specifies the sequence of colors to use when plotting multiple inputs on an analog graph.

& RDEV,(D#) ,(W#),(TV)[,(FU),(AM),(XM),(GA)/(GB)/(PR),(CV)]

Expansion command; used to read a value from any ISAAC expansion device.

& @RDEV,(D#),(W#),(AV) [,(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),
(RT),(SW)]

Matrix version of & RDEV.

& RETRCE

Re-starts a plot at the left margin of the currently defined graph without erasing previous plots.

& SCROLLSET

Initializes the scrolling graph format.

& SLOT# = aexpr

Apple peripheral slot # where the ISAAC/Apple Interface Card is located. The default value of aexpr is 3.

& TIME TO avar1,avar2,avar3

Reads the hour, minute, and second from the real-time clock.

& TIMERIN,(TV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV)]

Reads the current timer value.

& TRIGIN,(TV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV)]

Reads the current state of the Schmitt triggers.

& @TRIGIN,(AV) [(D#),(FU),(AM),(XM),(GA)/(GB)/(PR),(CV),(RT),(SW)]

Matrix version of & TRIGIN.

& TRIGPOLL,(TV) [(D#),(FU),(AM),(XM),(PR),(CV)]

Waits for one of the Schmitt triggers to be set, then returns the number of that trigger to (TV).

& @TRIGPOLL,(AV) [(D#),(FU),(AM),(XM),(PR),(CV),(RT),(SW)]

Matrix version of & TRIGPOLL.

& WRDEV,(D#),(W#),(DV) [(FU),(AM),(XM),(GA)/(GB)/(PR)]

Expansion command; used to write a value to any ISAAC expansion device.

& @WRDEV,(D#),(W#),(AV) [(FU),(AM),(XM),(GA)/(GB)/(PR)]

Matrix version of & WRDEV.

&&&

APPENDIX L: SOME USEFUL SUBROUTINES

This appendix contains a collection of subroutines written in Applesoft BASIC that you may find valuable when creating your own LabSoft programs.

The first group of these routines convert data from LabSoft's internal format to various other formats. These conversions were written primarily for use with the binary input commands. The binary data is input as one 16bit word. This is a convenient format for LabSoft's internal use, since it allows all of the binary data to be manipulated with a single command. However, depending on what is connected to those inputs, it may be more useful to represent the data in some other format.

L.1 Integer to Hexadecimal Conversion

ARGUMENTS: BIN (Value as input)
RETURNS: HEX\$ (Value converted to hex)

Here are a few sample conversions done with this subroutine.

BIN	HEX\$
10	\$000A
255	\$00FF
10828	\$2A4C

This routine takes the value in the variable BIN and converts it to hexadecimal format. The hex representation is returned in the string variable HEX\$. This routine makes internal use of two local variables (INDEX and TEMP).

```
100 REM CONVERT TO HEX
110 INDEX = 4096
120 HEX$ = "$"
130 FOR X = 3 TO 0 STEP -1
140 TEMP = INT (BIN / INDEX)
150 BIN = BIN - (INDEX * TEMP)
```

(PROGRAM CONTINUED ON NEXT PAGE)

```

160 INDEX = INDEX / 16
170 IF TEMP < 10 THEN HEX$ = HEX$ + STR$ (TEMP)
180 IF TEMP < 9 THEN HEX$ = HEX$ + CHR$ (55 + TEMP)
190 NEXT
200 RETURN

```

L.2 Convert Integer to Binary Format

ARGUMENTS: BIN (Value as input)
 RETURNS: BIN\$ (Value converted to binary format)

This routine takes the value in the variable BIN and converts it to a binary number. If you wish to look at the state of individual bits within the binary word, this type of conversion will be necessary. The binary representation is returned in the string variable BIN\$. This routine also uses the two local variables INDEX and TEMP.

Here are a few sample conversions done using this routine.

BIN	BIN\$
=====	=====
7	00000000 00000111
32	00000000 00010000
259	00000001 00000011
20000	01001110 00100000

```

100 REM CONVERT TO BINARY FORMAT
110 INDEX = 32768
120 BIN$ = ""
130 FOR X = 15 TO 0 STEP -1
140 IF X = 7 THEN BIN$ = BIN$ + " "
150 TEMP = BIN - INDEX
160 INDEX = INDEX / 2
170 IF TEMP < 0 THEN BIN$ = BIN$ + "0": GOTO 200
180 BIN$ = BIN$ + "1"
190 BIN = TEMP
200 NEXT
210 RETURN

```

L.3 Convert to ASCII

ARGUMENTS: BIN (Value as input)
 RETURNS: BIN\$ (Value converted to ASCII)

This routine takes the value in the variable BIN and converts it to two ASCII characters. The ASCII representation is returned in the string variable

BIN\$. The high order byte is converted to one character and the low order byte to another. Lower case letters are converted to upper case, and any non-printable or illegal characters are replaced with ".".

Some typical conversions done using this routine are shown in the following table.

BIN	ASCII Char.
83	S
33	!
43	+

```

100 REM CONVERT INTEGER TO ASCII
110 B1 = INT (BIN / 256)
120 B2 = BIN - (B1 * 256)
130 IF B1 > 127 THEN B1 = B1 - 128
140 IF B1 < 32 THEN B1 = 46
150 IF B2 > 127 THEN B2 = B2 - 128
160 IF B2 < 32 THEN B2 = 46
170 BIN$ = CHR$ (B1) + CHR$ (B2)
180 RETURN

```

The next few routines complement those above. These convert values entered in various formats to a 2byte integer. They were written primarily for use with the binary output commands.

L.4 Convert Hexadecimal to Integer

ARGUMENTS: BO\$ (4digit hex value as entered)
 RETURNS: BO (Converted output value)

E - Error flag (E=1, if there was an error)

This routine takes a hex value, represented as a 4character string (BO\$), and converts it to an integer (BO). If the string is longer than 4 characters or if there are any illegal values, then E is set = 1 and BO = 0.

Conversions done by this routine are the reverse of those done by routine L.1 above.

```

100 REM CONVERT FROM HEX
110 IF LEN (BO$) > 4 GOTO 190
120 FOR X = LEN (BO$) TO 1 STEP -1

```

```

130 TEMP = ASC (MID$ (BO$,X,1))
140 REM CHARACTER IS A NUMBER
150 IF TEMP > 47 AND TEMP < 58 THEN BO =BO + (TEMP - 48) * INDEX:
GOTO 200
160 REM CHARACTER IS A LETTER
170 IF TEMP > 63 AND TEMP < 71 THEN BO = BO + (TEMP - 55) *
INDEX: GOTO 200
180 REM NOT A VALID CHARACTER, SET ERROR FLAG
190 E=1
200 INDEX = INDEX * 16
210 NEXT
220 IF E = 1 THEN BO = 0
230 RETURN

```

L.5 Convert Binary to Decimal

ARGUMENTS: BO\$ (Value in binary format)
 RETURNS: BO (Converted value)

E Error flag (E=1 if there was an error)

This routine takes a value represented in binary form (BO\$) and converts it to an integer (BO) for use by the binary output command. The binary value can be entered as any number of bits between 1 and 16. The rightmost bit is considered the low order bit. Spaces may be added in the input (for clarity). If more than 16 bits are entered or if a bit is anything other than a "0," "1," or a " ", an error will be generated. An error is indicated by E = 1. This routine makes internal use of local variables BT, L, and TEMP\$.

Conversions done by this routine are the reverse of those done by routine L.2 above.

```

100 BT = 1 : L = 0 : BO = 1
110 E = 0
120 FOR X = LEN (BO$) TO 1 STEP -1
130 TEMP$ = MID$ (BO$,X,1)
140 IF TEMP$ = " " GOTO 210
150 IF TEMP$ = "1" THEN BO = BO + BT: GOTO 190
160 IF TEMP$ = "0" GOTO 190
170 REM IF DIGIT NOT 0, 1, OR NULL, EXIT WITH ERROR
180 E =1
190 BT = BT * 2
200 L = L + 1
210 NEXT
220 REM CHECK FOR RANGE ERROR
230 IF BO > 65535 THEN E = 1

```

```

240 IF L > 16 THEN E = 1
250 IF E = 1 THEN BO = 0
260 RETURN

```

L.6 Convert ASCII to Integer

ARGUMENTS: BO\$ (ASCII value to be converted)
 RETURNS: BO (Converted value)

E Error flag (E = 1 if error occurred)

This routine takes 2 characters in (BO\$) and converts them to their ASCII codes (BO). The low byte represents the rightmost character and the high byte represents the leftmost. If more than 2 characters are entered, the error flag will be set, and BO = 0.

Conversions done by this routine are the reverse of those done by routine L.3 above.

```

100 REM CONVERT ASCII TO INTEGER
110 E = 0
120 L = LEN (BO$)
130 REM IF MORE THAN 2 CHARACTERS, EXIT WITH ERROR
140 IF L > 2 THEN E = 1: GOTO 200
150 REM GET FIRST CHARACTER
160 BO = ASC (RIGHT$ (BO$,1))
170 IF L=1 GOTO 200
180 REM GET 2ND CHARACTER, IF THERE IS ONE
190 BO = BO + 256 * ASC (LEFT$ (BO$,1))
200 IF E=1 THEN BO = 0
210 RETURN

```

The next routine provides a method for getting at a hidden feature of the LabSoft & LABEL command. The character set provided with LabSoft contains both upper and lower case letters. Unfortunately the Apple converts all lower case letters to upper case automatically. This subroutine allows you to generate strings with both upper and lower case, for use with the & LABEL command.

L.7 String Conversion Routine

ARGUMENTS: OUT\$ (The string to be converted)
 RETURNS: OUT\$ (The converted string)

Send this routine a string (OUT\$) and it will be converted to lower case. Any character in the

original string that is preceded by the "up-arrow" character (^) will be left as upper case. All other letters will be converted to lower case. Numbers and symbols are left unchanged. This routine makes internal use of local variables X, Y, and TEMP\$. The converted string is returned in OUT\$. If you would rather not destroy the original string, you could return the converted string in some other variable.

```

90 REM CONVERT TO LOWER CASE
100 TEMP$ = "": UC = 0
110 FOR X = 1 TO LEN (OUT$)
120 REM IF CHAR. IS "^" THEN SET UPPER CASE FLAG
130 IF MID$ (OUT$,X,1) = "^" THEN UC = 1: GOTO 210
140 IF UC = 1 THEN TEMP$ = TEMP$ + MID$ (OUT$,X,1) : GOTO 200
150 Y = ASC ( MID$ (OUT$,X,1))
160 REM IF CHAR. IS NOT A LETTER THEN DON'T CONVERT
170 IF Y < 65 OR Y > 90 THEN Y = Y - 32
180 REM OTHERWISE CONVERT TO LOWER CASE
190 TEMP$ = TEMP$ + CHR$ (Y + 32)
200 UC = 0
210 NEXT
220 OUT$ = TEMP$
230 RETURN

```

An example may make things more clear. Let's say that you want to use the label command to print the string "Label" on the HIRES screen. Since you can't enter lower case characters from the Apple keyboard, it would be difficult to pass the appropriate characters to the & LABEL command. This utility can help. First set OUT\$ to the string you wish to use with "label." (Remember that all alpha characters except those preceded by the "up-arrow" will be converted to lower case). Then call the CONVERT TO LOWER CASE utility. Finally, use the converted out\$ as the argument in the & LABEL command as shown below.

```

2000 OUT$ = "^LABEL"
2010 GOSUB 100
2020 REM CONVERT STRING
2030 & LABEL = OUT$ AT 100, 110

```

&&&

APPENDIX M: OPTIMIZING THE USE OF INTEGER VARIABLES

Applesoft has two different types of arithmetic variables - integers and reals. They are handled and stored differently, and each has its own advantages and disadvantages. Both Applesoft and LabSoft were designed to make primary use of the real variables with which they perform all their arithmetic operations. However, both Applesoft and LabSoft also allow limited use of integer variables that can, at times, be advantageous.

The two key reasons for using integer variables are conservation of memory space and improvement of operation speed. Integers take up 2 bytes of memory per value, rather than the 5 bytes taken up by reals. In addition, all operations which use integers are significantly faster than those which use reals. This is true for LabSoft since all the values that are handled by ISAAC are integers to begin with. Storing these values as integer variables eliminates the time-consuming step of converting from integer to real or vice-versa.

Matrix commands are the principal LabSoft operations where considerations of space and speed become significant enough to warrant the use of integer variables.

The sole purpose of the matrix command is to do many I/O operations quickly. Since these commands handle large amounts of data, efficient use of memory can be important. Because the matrix command's primary advantage is speed, anything done to increase its speed will probably be an improvement. This is not true with simple commands, because the potential savings in execution speed are insignificant compared to the time it takes to interpret the command.

These space and time savings with integer variables come at a price. There are several important considerations involved with the use of integers. The first thing to consider is that all potential speed gains will be lost if the LabSoft command requires any arithmetic operation. If you specify

a conversion function (FU) or a compare value (CV), LabSoft will automatically convert the value from an integer to a real. Any use of integer variables here would be pointless. If you're not worried about speed in a particular application and you choose to use integer variables solely for the space savings, a conversion function (FU) can still get you into trouble. As mentioned above, all I/O values can be stored as integers. However, if you use a function with an I/O value, the result will not necessarily be within the integer value range. If it is not, an ?ILLEGAL QUANTITY ERROR will be generated.

The other problem with using integer variables occurs as a result of the way in which Applesoft handles integers internally. A 2byte integer can represent 65536 different values. Because the designers of Applesoft chose to have this range centered at zero, both positive and negative numbers could be represented, so that the range of allowable integer values in Applesoft is -32767 to +32767. This is generally a more useful range of numbers than 0 to 65535, but it can cause some confusion when dealing with 16-bit I/O.

The way Applesoft represents a negative integer is by setting the high order bit. In LabSoft however, the high order bit of an I/O value has nothing to do with negative numbers. This inconsistency about what the high order bit represents can easily create confusion.

For example, an & COUNTERIN command will read the (16bit) value of the counter into a target variable. If the first 15 bits of the counter are set, but the high order bit is cleared, this command will return the value 32767. The same value will be returned whether you specify an integer or a real as the target variable. Everything works just as you would expect. Now on the next cycle of the counter, the high order bit will be set, and all the others will be cleared. If you return this value to a real variable, it will read as 32768, just as expected. However, if you return it to an integer variable, it will read -32768. In general any LabSoft data value greater than 32767 that is stored in an integer variable will be converted by Applesoft into its negative equivalent.

You should be aware of the tendency for this kind of conversion to occur whenever you are dealing with a 16-bit LabSoft value expressed as an integer variable.

In addition to the little problem noted above, there are a few other quirks involving the use of integer variables in LabSoft I/O operations.

Analog inputs and analog outputs are all 12-bit values. When using & @AIN or & @AOUT commands the high order bit will never be set, so this won't be a problem. However, it is possible (and likely) that you will have difficulties when using the & @ASUM command with integers. The first few values will be summed without difficulty, but as you sum more values, they may eventually add up to more than 32767 and the value will be represented as a negative number. If you continue to sum more values, you will eventually get an ILLEGAL QUANTITY ERROR, since an integer value cannot be greater than 65535.

Binary, counter and timer inputs are all 16 bits wide. If you specify integer variables with these commands, they might return negative numbers. You can approach this problem in two ways. One way is to mask out the high order bit which is done by specifying an "AND" mask [as (AM)= 32767] in the command. However, if that last bit is important, you may store the values "as is" and process them later with the code shown below.

The following code goes through an integer array (INPUT%) and converts it to an equivalent real array (INPUT). All integer values that are less than 0 are converted to their positive equivalents.

```
100 FOR X = 1 TO 100
110 IF INPUT%(X) >= 0 THEN INPUT(X) = INPUT%(X): GOTO 130
120 INPUT(X) = INPUT%(X) + 65536
130 NEXT
```

&&&

APPENDIX U: UTILITY DISK PRINT-OUT

The following pages contain a print-out of the material contained on the LabSoft Utility Disk.

This material is presented exactly as it appears on the disk, and is therefore not formatted like the rest of this manual. It is here for your convenience, so that you may work with the Utility Disk information in written form rather than on your monitor. We hope this is useful to you.

LABSOFT UTILITY DISK
THIS DISK CONTAINS THE FOLLOWING UTILITY PROGRAMS:

THE SIGNAL SCREEN - A LABSOFT PROGRAM THAT DISPLAYS ALL OF ISAAC'S INPUTS AND OUTPUTS ON THE SCREEN. THIS PROGRAM INCLUDES TUTORIAL TYPE INSTRUCTIONS THAT DESCRIBE SOME OF THE FEATURES OF LABSOFT. TO RUN THE PROGRAM, FIRST "BOOT" THE LABSOFT MASTER DISK. ONCE LABSOFT IS LOADED, INSERT THE UTILITY DISK AND TYPE "RUN SIGNAL SCREEN."

THE GRAPH FORMATTER - A GRAPHICS UTILITY THAT LABELS LABSOFT GRAPHS. IN A FEW MINUTES YOU CAN SET UP AND LABEL A LABSOFT GRAPH AND THEN SAVE IT ON A DISK. FOR A FURTHER DESCRIPTION AND INSTRUCTIONS, TYPE "RUN GRAPH FORMATTER."

THE TRANSIENT RECORDER - AN ASSEMBLY LANGUAGE PROGRAM FOR RECORDING VERY FAST TRANSIENT EVENTS AT ISAAC'S ANALOG INPUT. IT WAITS FOR A TRIGGER AND THEN SAMPLES FROM ONE ANALOG CHANNEL AT SPEEDS OF UP TO 8.5 KHZ. FOR MORE INFORMATION, TYPE "RUN TRANSIENT INSTRUCTIONS."

THE PRE-TRIGGER RECORDER - AN ASSEMBLY LANGUAGE PROGRAM SIMILAR TO THE TRANSIENT RECORDER EXCEPT THAT THIS ONE RECORDS SAMPLES BOTH BEFORE AND AFTER THE TRIGGER. IT'S TOP SAMPLING SPEED IS ABOUT 6.8 KHZ. FOR MORE INFORMATION, TYPE "RUN PRE-TRIGGER INSTRUCTIONS."

APPLE SLICER - AN ASSEMBLY LANGUAGE ROUTINE THAT WILL SPLIT ANY BASIC OR LABSOFT PROGRAM IN HALF, SO THAT IT WILL "WRAP AROUND" THE APPLE'S GRAPHICS MEMORY. IN NORMAL USE, PROGRAMS THAT USE THE APPLE'S HI-RES SCREENS WASTE A LOT OF MEMORY. THE APPLE SLICER CAN RECOVER A LOT OF THAT WASTED SPACE FOR YOU. FOR MORE DETAILS, TYPE "RUN APPLE SLICER INSTRUCTIONS."

SIGNAL SCREEN INSTRUCTIONS

THE SIGNAL SCREEN IS A VERY GENERAL PURPOSE UTILITY PROGRAM THAT EXERCISES THE VARIOUS INPUTS AND OUTPUTS OF THE ISAAC SYSTEM.

IT ALLOWS YOU TO SPECIFY THE BINARY AND ANALOG OUTPUTS FOR ALL AVAILABLE CHANNELS, IN A VARIETY OF FORMATS. THEN IT DISPLAYS THIS INFORMATION ALONG WITH ALL OF THE BINARY, ANALOG, TRIGGER AND COUNTER INPUTS, ON A SINGLE SCREEN.

SINCE THE SIGNAL SCREEN DISPLAYS THE STATUS OF ALL THE INPUTS AND OUTPUTS SIMULTANEOUSLY, IT PROVIDES AN EASY MEANS OF CHECKING OUT THE ENTIRE SYSTEM.

IT IS ALSO A QUICK AND EASY WAY TO USE THE SYSTEM, WITHOUT WRITING A LABSOFT PROGRAM.

UPON RUNNING THE PROGRAM, THE FIRST MENU THAT IS DISPLAYED ALLOWS YOU TO SPECIFY A CONVERSION FUNCTION FOR THE BINARY OR ANALOG INPUTS OR OUTPUTS.

- A. A/D RANGE IS -5 TO +5 VOLTS
- B. D/A RANGE IS -5 TO +5 VOLTS
- C. BINARY INPUT FORMAT IS DECIMAL
- D. BINARY OUTPUT FORMAT IS DECIMAL

E. NO CHANGES

THE FIRST TWO MENU CHOICES ALLOW YOU TO SPECIFY A CONVERSION FUNCTION FOR EITHER THE D/A OR A/D CONVERTERS. TO UNDERSTAND HOW TO USE THIS, YOU MUST FIRST KNOW WHAT D/A AND A/D CONVERTERS DO.

THE COMPUTER HANDLES ALL INFORMATION IN A DIGITAL FORM - AS JUST A NUMBER.

A D/A CONVERTER TAKES THIS NUMBER AND CONVERTS IT TO A VOLTAGE.

AN A/D CONVERTER TAKES A VOLTAGE AND CONVERTS IT TO A NUMBER

THE PARTICULAR A/D'S AND D/A'S USED IN THIS SYSTEM ARE 12 BIT DEVICES AND THEIR STANDARD RANGE IS -5 TO +5 VOLTS.

WITH 12 BITS YOU CAN REPRESENT ANY NUMBER IN THE RANGE OF 0 - 4095.

SO IN THIS SYSTEM, IF YOU DO NOT USE A CONVERSION FUNCTION, THEN -

A COUNT OF 0 = -5 VOLTS AND
A COUNT OF 4095 = +5 VOLTS AND

THERE IS A LINEAR RELATIONSHIP FOR ALL THE VALUES IN BETWEEN, THAT

IS:

$$\text{COUNT} = (\text{VOLTS} + 5) * 409.5$$

THIS IS THE CONVERSION FUNCTION OF THE ACTUAL A/D AND D/A
HARDWARE.

GRAPHICALLY IT LOOKS LIKE THIS -

DIGITAL	>>>	(D/A)	>>>	ANALOG
0 - 4095		>>>		-5 TO +5 V

ANALOG	>>>	(A/D)	>>>	DIGITAL
-5 TO +5 V		>>>		0 - 4095

HOWEVER, OFTEN IT'S USEFUL TO CONVERT THE RAW VALUES TO SOME OTHER
RANGE OR TYPE OF UNITS. THAT IS JUST WHAT THE OPTIONAL CONVERSION
FUNCTION IS FOR.

WHEN YOU FIRST RUN THE SIGNAL SCREEN, IT IS AUTOMATICALLY SET UP
FOR A CONVERSION FUNCTION ON BOTH THE ANALOG INPUT AND OUTPUTS.

THIS PRESET CONVERSION FUNCTION, IS JUST A MIRROR IMAGE OF THE A/D
AND D/A'S INTERNAL CONVERSION.

BY USING THIS CONVERSION, YOU WON'T HAVE TO CONCERN YOURSELF WITH
THE DIGITAL VALUES (0 - 4095) THAT THE CONVERTERS USE.

YOU CAN SPECIFY A +5 VOLT OUTPUT. THE CONVERSION FUNCTION WILL
TRANSLATE THIS INTO THE NUMBER 4095. THEN THE A/D CONVERTER WILL
GENERATE A +5 V OUTPUT.

GRAPHICALLY, THIS CONVERSION LOOKS LIKE THIS:

IN	FUNCTION		A/D		OUT
+5 V	>>>	4095	>>>		+5V

AND THE SAME CONVERSION FOR THE D/A WOULD LOOK LIKE THIS:

IN	D/A		FUNCTION		OUT
+5 V	>>>	4095	>>>		+5V

USING THE STANDARD CONVERSION IS ALMOST ALWAYS EASIER THAN DEALING
WITH JUST THE RAW DIGITAL VALUES. HOWEVER IF YOU HAVE SOME
EXTERNAL DEVICE CONNECTED TO ONE OF THE ANALOG INPUTS OR OUTPUTS,
IT IS USUALLY EVEN EASIER TO DEFINE YOUR OWN CONVERSION.

ONCE YOU HAVE CORRECTLY DEFINED THE CONVERSION, YOU WILL ONLY HAVE
TO THINK IN TERMS OF THE UNITS THAT YOU ARE MEASURING RATHER THAN
A/D COUNTS OR VOLTAGES.

FOR EXAMPLE: YOU HAVE AN ELECTRONIC THERMOMETER THAT HAS A 0-5 VOLT OUTPUT, AND YOU WANT TO CONNECT IT TO THE ISAAC. THE THERMOMETER GENERATES AN OUTPUT OF 0 VOLTS FOR 0 DEGREES F AND 5 VOLTS FOR 100 DEGREES F. THE SYSTEM WILL LOOK LIKE THIS:

THERMOMETER		(A/D)
DEGREES	>>> VOLTS	>>> A/D COUNT
0	0	2048
100	5	4095

THE THERMOMETER CONVERTS A TEMPERATURE TO A VOLTAGE. THEN THE A/D CONVERTS THIS VOLTAGE TO A DIGITAL NUMBER. WITH THE SYSTEM SET UP LIKE THIS, YOU CAN CALCULATE THE OUTPUT FOR ANY TEMPERATURE INPUT. HOWEVER, RATHER THAN DOING THIS CALCULATION EVERY TIME YOU MAKE A MEASUREMENT, YOU CAN DEFINE THIS CONVERSION, AND LET THE COMPUTER DO IT FOR YOU.

- A. A/D RANGE IS -5 TO +5 VOLTS
- B. D/A RANGE IS -5 TO +5 VOLTS
- C. BINARY INPUT FORMAT IS DECIMAL
- D. BINARY OUTPUT FORMAT IS DECIMAL

E. NO CHANGES.

TO DEFINE THIS CONVERSION, YOU WOULD SELECT (A) FROM THE MENU.

WHICH CONVERSION WOULD YOU LIKE ?

CONVERT THE RAW A/D VALUES
(0 - 4095) TO -

- A. -5 TO +5 VOLTS
- B. -10 TO +10 VOLTS
- C. DEFINE YOUR OWN CONVERSION
- D. NO CONVERSION

FOR OUR EXAMPLE, YOU WOULD SELECT C. DEFINE YOUR OWN CONVERSION.

INP

WHAT UNITS DO YOU WANT THE RAW A/D VALUES CONVERTED TO ?

FOR THIS PARTICULAR CONVERSION, YOU WOULD ENTER DEGREES, SINCE THAT IS THE TYPE OF UNITS THAT ARE ACTUALLY BEING MEASURED.

AN A/D COUNT OF 0 = HOW MANY DEGREES ?

THIS NEXT SECTION IS THE CRITICAL PART OF DEFINING THE CONVERSION, AND IT CAN BE A LITTLE CONFUSING.

THE THERMOMETER IN THE EXAMPLE HAS A 0-5 VOLT OUTPUT FOR A 0-100 DEGREE INPUT. THE A/D CONVERTER ACCEPTS A WIDER RANGE OF VOLTAGES (-5 TO +5 VOLTS), THAN THE THERMOMETER WILL GENERATE. TO ANSWER

THE QUESTION ABOVE, YOU MUST CALCULATE THE TEMPERATURE THAT WOULD GIVE A THERMOMETER OUTPUT OF -5 VOLTS (A/D COUNT=0), IF THE THERMOMETER COULD MEASURE THAT LOW.

AN A/D COUNT OF 0 = HOW MANY DEGREES ?

IF FOR THIS THERMOMETER:

0 DEGREES IN GIVES 0 VOLTS OUT AND
100 DEGREES IN GIVES 5 VOLTS OUT

THEN:

-100 DEGREES IN GIVES -5 VOLTS OUT

SO YOU WOULD ANSWER -100.

AN A/D COUNT OF 4095=HOW MANY DEGREES?

THE ANSWER TO THIS ONE IS PROBABLY A LITTLE CLEARER. AN A/D COUNT OF 4095 = +5 VOLTS. THE THERMOMETER WILL GENERATE A +5 VOLT OUTPUT FOR +100 DEGREE INPUT. SO, YOU SHOULD ANSWER 100 TO THIS QUESTION.

ONCE YOU'VE ANSWERED THESE QUESTIONS, THE CONVERSION IS COMPLETELY DEFINED AND YOU WILL RETURN BACK TO THE ORIGINAL MENU.

SIGNAL SCREEN INSTRUCTIONS!

- A. A/D RANGE IS -100 TO +100 DEGREES
- B. D/A RANGE IS -5 TO +5 VOLTS
- C. BINARY INPUT FORMAT IS DECIMAL
- D. BINARY OUTPUT FORMAT IS DECIMAL

E. NO CHANGES

NOTICE THAT (A) ON THE MENU, NOW DESCRIBES THE NEW CONVERSION THAT YOU JUST DEFINED. FROM THIS POINT ON, ALL A/D MEASUREMENTS WILL BE CONVERTED TO DEGREES, USING THE DEFINED CONVERSION.

THE SAME TYPE OF CONVERSION CAN ALSO BE DEFINED FOR THE D/A OUTPUTS AND THE PROCEDURE IS IDENTICAL.

THE SIGNAL SCREEN ALSO ALLOWS YOU TO DO CONVERSIONS ON THE BINARY INPUTS AND OUTPUTS. CHOICES (C) AND (D) ON THE MENU DISPLAY THE CURRENT FORMAT. CHOOSING EITHER OF THESE WILL ALLOW YOU TO SELECT A DIFFERENT FORMAT CONVERSION.

THE CONVERSIONS FOR THE BINARY INPUTS AND OUTPUTS ARE A DIFFERENT TYPE OF CONVERSION THEN THOSE JUST DESCRIBED FOR THE ANALOG SIGNALS.

TO UNDERSTAND THEIR PURPOSE, LET'S TAKE A CLOSER LOOK AT THE BINARY I/O AND ITS POTENTIAL APPLICATIONS.

THE BINARY INPUTS AND OUTPUTS ARE 16 BITS WIDE. EACH OF THE 16 LINES IS AVAILABLE TO THE USER AND EACH OF THESE 16 BITS CAN BE READ AND WRITTEN TO. BY ITS NATURE THIS KIND OF I/O IS VERY VERSATILE.

YOU COULD USE THESE BINARY LINES IN A NUMBER OF DIFFERENT WAYS. YOU CAN USE EACH OF THESE LINES SEPARATELY, TO MONITOR THE STATE OF 16 DIFFERENT SWITCHES. YOU COULD USE THE LINES AS 4 4-BIT WORDS, WHERE EACH WORD REPRESENTS A BCD DIGIT. OR YOU COULD BREAK IT INTO 2 8-BIT WORDS AND COMMUNICATE WITH A PRINTER OR SOME OTHER ASCII DEVICE.

THE SELECTABLE BINARY FORMATS ALLOW YOU TO DO ALL OF THESE THINGS EASILY.

WHEN YOU FIRST RUN THE SIGNAL SCREEN, IT IS SET UP FOR DECIMAL FORMAT. THIS IS THE NORMAL WAY IN WHICH THE MACHINE HANDLES THE BINARY INPUT AND OUTPUT VALUES. THIS FORMAT TREATS THE WHOLE BINARY WORD AS 1 16-BIT INTEGER VALUE.

THIS IS THE MOST GENERAL TYPE OF FORMAT, AND REQUIRES NOTHING MORE THAN A &BIN OR A &BOUT COMMAND TO IMPLEMENT. HOWEVER, THIS IS NOT USUALLY THE MOST CONVENIENT FORMAT TO USE.

SIGNAL SCREEN INSTRUCTIONS!

- A. A/D RANGE IS -100 TO +100 DEGREES
- B. D/A RANGE IS -5 TO +5 VOLTS
- C. BINARY INPUT FORMAT IS DECIMAL
- D. BINARY OUTPUT FORMAT IS DECIMAL

E. NO CHANGES

TO SELECT A NEW FORMAT, OR TO FIND OUT WHAT TYPE OF FORMATS ARE AVAILABLE, JUST SELECT EITHER (C) OR (D) FROM THE MENU.

SIGNAL SCREEN INSTRUCTIONS

WHICH FORMAT WOULD YOU LIKE ?

- A. BINARY (EX. 00100110 11001001)
- B. DECIMAL (EX. 34562)
- C. HEX (EX. \$34CB)
- D. ASCII (EX. PQ)
- E. BCD (EX. 9384)!

THESE ARE THE FIVE AVAILABLE FORMATS. YOU CAN USE THEM ON THE BINARY INPUT, OUTPUT, OR BOTH.

THE BINARY FORMAT (A) IS USEFUL IF YOU ARE INTERESTED IN THE STATE OF SINGLE BITS. IT ALLOWS YOU TO EASILY READ OR WRITE ANY OR ALL OF THE 16 BITS.

THE DECIMAL FORMAT (B) IS LABSOFT'S DEFAULT FORMAT FOR THE BINARY I/O. IT IS OFTEN AWKWARD TO USE BUT IT DOESN'T REQUIRE A CONVERSION.

THE HEX FORMAT (C) IS A STANDARD FORMAT FOR HANDLING BINARY NUMBERS. THIS FORMAT TREATS THE 16 BITS AS 4 4-BIT WORDS. EACH 4-BIT WORD CAN HAVE A VALUE FROM 0-9 OR A-F. (A-F REPRESENTS 10-15).

THE ASCII FORMAT IS TWO 8 BIT WIDE WORDS. THIS FORMAT IS A STANDARD WAY OF COMMUNICATING WITH PERIPHERAL DEVICES, ESPECIALLY ALPHANUMERIC TYPES (PRINTERS KEYBOARDS ETC.)

BCD FORMAT IS MOST OFTEN USED WITH EXTERNAL NON-COMPUTER DIGITAL DEVICES. THIS FORMAT BREAKS DOWN INTO 4 4-BIT WORDS. EACH WORD REPRESENTS A DECIMAL DIGIT (0-9). THOUGH IT IS POSSIBLE TO REPRESENT NUMBERS GREATER THAN 9 WITH 4 BITS, IT IS NOT ALLOWED IN THE BCD FORMAT.

AFTER SELECTING A NEW FORMAT, YOU WILL ONCE AGAIN BE RETURNED TO THE MENU FROM WHENCE YOU CAME.

SIGNAL SCREEN INSTRUCTIONS!

- A. A/D RANGE IS -100 TO +100 DEGREES
- B. D/A RANGE IS -5 TO +5 VOLTS
- C. BINARY INPUT FORMAT IS HEX
- D. BINARY OUTPUT FORMAT IS DECIMAL

E. NO CHANGES

AFTER ALL THE FORMATS ARE SET UP PROPERLY, YOU CAN THEN SELECT (E) FROM THE MENU, TO MOVE ON TO THE NEXT SECTION OF THE PROGRAM.

SIGNAL SCREEN INSTRUCTIONS

- A. D/A CHANNEL #0 = 0 VOLTS
- B. D/A CHANNEL #1 = 0 VOLTS
- C. D/A CHANNEL #2 = 0 VOLTS
- D. D/A CHANNEL #3 = 0 VOLTS
- E. BINARY OUT VALUE = 0 (DECIMAL)
- F. COUNTER CHANNEL # = 0
- G. CHANGE DISPLAY FORMAT
- H. NO CHANGES!

NOW YOU CAN SPECIFY THE VALUE OF ANY OR ALL OUTPUTS. JUST SELECT THE APPROPRIATE LETTER FROM THE MENU. (A-E).

WHEN YOU ENTER AN OUTPUT VALUE, BE SURE THAT IT IS IN THE FORMAT THAT YOU PREVIOUSLY SPECIFIED. I.E. DON'T ENTER THE VALUE 45C3 FOR THE BINARY OUTPUT, IF THE FORMAT IS SUPPOSED TO BE DECIMAL.

THE SIGNAL SCREEN ALSO READS THE FREQUENCY (IN KHZ) OF A SIGNAL AT ONE OF THE ISAAC COUNTER INPUTS. THIS COUNTER HAS EIGHT CHANNELS, BUT IT CAN ONLY COUNT ONE CHANNEL AT A TIME. CHOOSE (F) FROM THE MENU TO CLEAR THE COUNTER AND SELECT A CHANNEL.

IF YOU WANT TO GO BACK AND MAKE ANY CHANGES TO THE I/O FORMATS OR RANGES, SELECT (G), OTHERWISE SELECT (H), AND THE SIGNAL SCREEN WILL GO INTO OPERATION.

SIGNAL SCREEN INSTRUCTIONS!

ONCE YOU START IT RUNNING, IT WILL JUST CONTINUOUSLY LOOP THROUGH THE PROGRAM. ON EACH LOOP, THE SIGNAL SCREEN UTILITY WILL WRITE ALL OF THE VALUES THAT YOU SPECIFIED TO THE OUTPUT DEVICES. THEN IT WILL READ ALL OF THE INPUTS AND FORMAT THEM JUST AS YOU REQUESTED.

NOTE THAT ALL SIXTEEN OF THE ANALOG INPUTS WILL BE DISPLAYED ON THE SCREEN AT THE SAME TIME. IF YOU DON'T HAVE ANYTHING CONNECTED TO SOME OF THE INPUTS, THEY WILL PROBABLY NOT READ ZERO, THEY JUST 'FLOAT'. THE FLOATING INPUTS WILL TEND TO FOLLOW AN INPUT THAT IS CONNECTED TO SOMETHING. THIS CAN LOOK CONFUSING BUT IT IS PERFECTLY NORMAL.

AT ANY TIME, YOU CAN HIT THE KEYBOARD TO STOP THE DISPLAY. WHEN THE SIGNAL SCREEN'S DISPLAY STOPS YOU CAN THEN GO BACK AND CHANGE ANY OF THE OUTPUT VALUES OR FORMATS.

THAT'S IT FOR INSTRUCTIONS, SO HIT THE (N) KEY ONE MORE TIME, AND GIVE IT A TRY.

GRAPH LABELING INSTRUCTIONS

LABSOFT HAS A WHOLE SET OF COMMANDS THAT ALLOW YOU TO DO REAL TIME PLOTTING OF DATA POINTS ON EITHER OF THE HI-RES SCREENS.

THIS UTILITY ALLOWS YOU TO PREPARE THE OUTLINE AND LABELING FOR ONE OF THESE GRAPHS AND STORE IT ON A DISK.

WHEN YOU ARE READY TO RUN A LABSOFT PLOTTING PROGRAM, YOU CAN JUST LOAD IN ONE OF THESE PREVIOUSLY PREPARED SCREENS INTO EITHER OF THE HI-RES AREAS, AS A BACKDROP FOR THE LABSOFT PLOT.

THE FIRST QUESTION TO BE ANSWERED IS:

WHICH SCREEN WOULD YOU LIKE TO CREATE THE GRAPH ON ?

AFTER CREATING THIS GRAPH, IT WILL BE STORED ON A DISK AS A BINARY FILE.

WHEN YOU LATER LOAD THE GRAPH BACK INTO THE MACHINE, IT WILL BE LOADED INTO THE HI-RES SCREEN AREA THAT YOU SPECIFY NOW.

FOR MORE INFORMATION ON THE HI-RES SCREENS REFER TO "THE APPLE REFERENCE MANUAL"

GRAPH LABELING INSTRUCTIONS!

NEXT YOU WILL BE PRESENTED WITH THE MAIN MENU, WHICH LISTS ALL THE POSSIBLE COMMANDS OF THIS UTILITY PROGRAM.

GRAPH LABELING INSTRUCTIONS

- A. SELECT TYPE OF GRAPH
- B. SELECT TITLE OF GRAPH
- C. LABEL THE MAX
- D. LABEL THE MIN
- E. LABEL THE VERTICAL AXIS
- F. LABEL THE HORIZONTAL AXIS
- G. ADD TIC MARKS TO GRAPH
- H. ADD TEXT TO BOTTOM OF GRAPH
- I. DISPLAY THE GRAPH
- J. SAVE GRAPH ON DISK
- K. QUIT!

YOU CAN SELECT ANY OF THE CHOICES ON THE MENU IN ANY ORDER THAT YOU WISH, ALTHOUGH YOU OBVIOUSLY WOULDN'T WANT TO SELECT J. SAVE GRAPH ON DISK OR K. QUIT UNTIL AFTER YOU HAD FINISHED SETTING UP THE GRAPH.

THE SIMPLEST WAY TO CREATE A GRAPH IS TO SELECT A - H IN ORDER, SKIPPING ANY OF THE CHOICES THAT YOU AREN'T INTERESTED IN.

AFTER ADDING ANYTHING TO THE GRAPH, YOU CAN SELECT I. DISPLAY THE

GRAPH, TO SEE HOW IT LOOKS.

IF YOU WANT TO MAKE ANY CHANGES, JUST MAKE THE APPROPRIATE SELECTION FROM THE MENU AND MAKE A NEW ENTRY.

IF YOU WOULD LIKE TO ELIMINATE A LABEL OR A TITLE JUST MAKE THE APPROPRIATE SELECTION FROM THE MENU AND INSTEAD OF ENTERING A NEW LABEL OR TITLE, JUST HIT THE (RETURN) KEY.

WHEN THE GRAPH IS COMPLETE, SELECT J. SAVE GRAPH TO DISK AND THEN ENTER A NAME FOR THE GRAPH.

ONCE THE GRAPH IS STORED, YOU CAN SELECT K. QUIT

GRAPH LABELING INSTRUCTIONS!

THE ONLY ASPECT OF ALL THIS, THAT MIGHT EASILY CAUSE SOME CONFUSION, IS THAT YOU MUST USE THE GRAPHS THAT YOU CREATE IN THE WAY THAT YOU SPECIFIED WHEN YOU CREATED THEM.

MORE SPECIFICALLY - IF YOU SPECIFIED THAT THE GRAPH WAS FOR HI-RES SCREEN #2, THEN WHEN YOU USE IT YOU MUST PLOT THE POINTS ON SCREEN #2 (&HIRES2).

IF YOU ENTERED TEXT UNDER THE GRAPH (OTHER THAN THE LABELS), WHEN YOU USE IT YOU MUST SPECIFY A FULL SCREEN FORMAT (&FULLSCREEN) OR THE TEXT WILL NOT APPEAR.

IF YOU SPECIFIED A SPLIT SCREEN ALTERNATING GRAPH, THEN WHEN YOU USE IT YOU MUST SPECIFY THE SAME TYPE OF PLOT (&ALTSET).

THE TRANSIENT RECORDER

THE TRANSIENT RECORDER IS AN ASSEMBLY LANGUAGE PROGRAM THAT WAITS FOR A TRIGGER AND THEN FILLS UP A BASIC ARRAY WITH A/D SAMPLES.

THE POTENTIAL SAMPLING RATE IS SIGNIFICANTLY HIGHER THAN YOU CAN OBTAIN USING LABSOFT. THIS IS BECAUSE IT IS A SPECIAL PURPOSE PROGRAM THAT HAS ONLY ONE SIMPLE JOB TO DO AND DOESN'T HAVE TO HAVE ALL OF THE SOFTWARE "OVERHEAD" OF LABSOFT. THE DRAWBACK OF THIS TYPE OF PROGRAM IS THAT IT'S A LITTLE MORE COMPLICATED TO USE AND IT IS NOT VERY FLEXIBLE.

THE TRANSIENT RECORDER IS A RELOCATABLE PROGRAM. THIS MEANS THAT YOU CAN LOAD AND RUN IT ANYWHERE IN THE APPLE'S AVAILABLE USER MEMORY AND IT WILL WORK PROPERLY.

THE SIMPLEST PLACE TO PUT IT, IS USUALLY AT THE TOP OF THE AVAILABLE MEMORY. TO DO THIS -

START YOUR PROGRAM THIS WAY.

```
100 D$=CHR$(4)
110:
120 REM X=CURRENT VALUE OF HIMEM
130 X = PEEK (116) * 256 + PEEK(115)
140:
150 REM ADR=ADDRESS THAT TRANSIENT          RECORDER WILL BE LOADED
    AT
160 ADR=X-285
170:
180 REM LOAD TRANSIENT RECORDER
190 PRINT D$;"BLOAD TRANSIENT.OBJ,A";ADR
200:
210 REM SET HIMEM BELOW THE TRANSIENT      RECORDER SO THAT
220 REM BASIC DOES NOT WRITE OVER IT.
230 HIMEM:ADR
```

IF LABSOFT IS IN THE MACHINE, THIS CODE WILL LOAD THE TRANSIENT RECORDER BELOW IT. IF LABSOFT IS NOT IN THE MACHINE, IT WILL LOAD THE TRANSIENT RECORDER BELOW DOS.

TO USE THIS UTILITY, YOU MUST FIRST DIMENSION AN INTEGER ARRAY IN BASIC. THEN YOU CALL THE ROUTINE WITH A LIST OF PARAMETERS. THE FORMAT OF THE CALL STATEMENT IS

CALL STARTING ADDRESS OF ROUTINE, ARRAY NAME, SAMPLING RATE,
ANALOG CHANNEL#, TRIGGER VALUE, INTERFACE CARD SLOT# (OPTIONAL)

ALL BUT THE LAST PARAMETER ARE REQUIRED.

TRANSIENT RECORDER PARAMETERS

STARTING ADDRESS OF ROUTINE

THIS IS ALWAYS THE ADDRESS THAT THE TRANSIENT RECORDER WAS LOADED AT.

ARRAY NAME

THIS IS THE NAME OF THE ARRAY THAT THE TRANSIENT RECORDER WILL STORE IT'S DATA IN. IT MUST BE AN INTEGER ARRAY AND IT MUST ALSO HAVE BEEN PREVIOUSLY DIMENSIONED. IF THE ARRAY THAT IS SPECIFIED HERE IS NOT INTEGER OR IF IT HASN'T BEEN PREVIOUSLY DIMENSIONED, YOU WILL GET AN OUT OF DATA ERROR.

SAMPLING RATE

THIS IS SIMILIAR TO THE SAMPLING RATE PARAMETER IN LABSOFT (RT). IT MUST BE A NUMBER IN THE RANGE 0 - 255. SPECIFYING THE SAMPLE RATE AS 0 WILL CAUSE THE ROUTINE TO ACQUIRE DATA AS FAST AS IT CAN (FREE RUNNING). THE FREE RUNNING SPEED IS APPROX 118 US/SAMPLE (8.5 KHZ). EVERY UNIT ADDED TO THE SAMPLING RATE, ADDS ANOTHER 10 US PER SAMPLE. UNLIKE LABSOFT, THIS ROUTINE'S TIMING IS NOT DONE WITH A HARDWARE TIMER AND IT IS NOT NEARLY AS ACCURATE NOR AS STABLE. THE TIMES LISTED BELOW ARE APPROXIMATIONS, AND THERE WILL BE SLIGHT SAMPLE TO SAMPLE VARIATIONS.

SAMPLE SPEED =

118 US + (SAMPLING RATE * 10 US)

SAMPLE RATE

PARAMETER	SAMPLING SPEED
255 -----	2668 US (374 HZ)
100 -----	1118 US (894 HZ)
10 -----	218 US (4.6 KHZ)
1 -----	130 US (7.7 KHZ)
0 -----	118 US (8.5 KHZ)

ANALOG CHANNEL

THIS IS THE A/D CHANNEL# FROM WHICH YOU WISH TO TAKE THE SAMPLES. MUST BE IN THE RANGE 0-15.

TRIGGER VALUE

THIS PARAMETER SPECIFIES WHICH SCHMITT TRIGGER INPUT(S) WILL BE USED TO START THE SAMPLING. THE TRIGGER VALUE SHOULD BE EITHER 1,2,4,OR 8 DEPENDING ON WHICH INPUT YOU CHOOSE TO USE.

INTERFACE CARD SLOT#

THE RECOMMENDED LOCATION FOR THE ISAAC INTERFACE CARD IS IN APPLE PERIPHERAL SLOT #3. IF YOUR CARD IS IN THIS SLOT THEN YOU DON'T HAVE TO SPECIFY THIS PARAMETER. IF IT IS LOCATED IN ANY OTHER APPLE PERIPHERAL SLOT, THEN YOU MUST ENTER THE SLOT# AS THE LAST PARAMETER IN THE COMMAND.

UNLIKE LABSOFT, THESE PARAMETERS MUST APPEAR IN THE SPECIFIC ORDER DESCRIBED ABOVE.

THE TRANSIENT RECORDER

IF WE WERE TO CONTINUE WITH THE PROGRAM STARTED ABOVE, THE NEXT LINES MIGHT BE

```
240:
250 REM DIMENSION AN AN INTEGER ARRAY
260 DIM AD%(5000)
270:
280 REM EXECUTE THE TRANSIENT RECORDER PROGRAM
290 CALL ADR,AD%,0,7,1
```

IN THIS EXAMPLE WE FIRST DIMENSION AN ARRAY WITH 5001 ELEMENTS. THEN WE CALL THE TRANSIENT RECORDER UTILITY WITH A LIST OF PARAMETERS.

!NP

THE FIRST PARAMETER IS THE STARTING ADDRESS OF THE ROUTINE WHICH WAS STORED AS ADR ABOVE. THE NEXT PARAMETER IS THE ARRAY NAME. IN THIS EXAMPLE IT'S AD%. NEXT IS THE SAMPLING RATE, WHICH IS SET TO 0. THIS MEANS THAT THE ROUTINE WILL SAMPLE AS FAST AS IT CAN (APROX. 8.5 KHZ). THE A/D CHANNEL#=7 AND THE TRIGGER VALUE=1.

WHAT THIS ROUTINE WILL DO IN THIS EXAMPLE, IS CONTINUOUSLY MONITOR SCHMITT TRIGGER INPUT #0. AS SOON AS THE THRESHOLD CONDITION OF THE TRIGGER IS MET, THE ROUTINE WILL TAKE 5001 SAMPLES FROM A/D CHANNEL #7 AND STORE THEM IN THE ARRAY AD% AS FAST AS IT CAN.

PRE-TRIGGER RECORDER

OR

COUNTING YOUR CHICKENS BEFORE THEY HATCH

PRE-TRIGGER RECORDER

THIS IS AN ASSEMBLY LANGUAGE PROGRAM THAT FILLS A BASIC INTEGER ARRAY WITH A/D SAMPLES, MUCH LIKE THE TRANSIENT RECORDER. HOWEVER, THIS PROGRAM ALLOWS YOU TO SPECIFY HOW MANY POINTS ARE TO BE TAKEN BEFORE A TRIGGER AND HOW MANY AFTER. THIS CAN BE PARTICULARLY USEFUL IF YOU WANT TO LOOK AT THE RISING EDGE OF A WAVEFORM, SINCE YOU CAN EASILY GET A TRIGGER FROM THE PEAK OF THE WAVEFORM BUT STILL HAVE A RECORD OF THE A/D VALUES PRECEDING THE TRIGGER. THIS PROGRAM (LIKE THE TRANSIENT RECORDER) IS CAPABLE OF SAMPLING AT A HIGHER SPEED THAN LABSOFT, IF YOU DON'T MIND SACRIFICING LABSOFT'S FLEXIBILITY AND EASE OF USE.

THE PRE-TRIGGER RECORDER IS A RELOCATABLE PROGRAM. THIS MEANS THAT YOU CAN LOAD AND RUN IT ANYWHERE IN THE APPLE'S AVAILABLE USER MEMORY AND IT WILL WORK PROPERLY. THE SIMPLEST PLACE TO PUT IT, IS USUALLY AT THE TOP OF THE AVAILABLE MEMORY. TO DO THIS -

START YOUR PROGRAM THIS WAY

```
100 D$=CHR$(4)
110:
120 REM X=CURRENT VALUE OF HIMEM
130 X = PEEK (116) * 256 + PEEK(115)
140:
150 REM ADR=ADDRESS THAT PRE-TRIGGER REC      ORDER WILL BE LOADED
    AT
160 ADR=X-690
170:
180 REM LOAD PRE-TRIGGER RECORDER
190 PRINT D$;"BLOAD PRE-TRIGGER.OBJ,A";A      DR
200:
210 REM SET HIMEM BELOW THE PRE-TRIGGER      RECORDER SO THAT
220 REM BASIC DOES NOT WRITE OVER IT.
230 HIMEM:ADR
```

IF LABSOFT IS IN THE MACHINE, THIS CODE WILL LOAD THE PRE-TRIGGER RECORDER BELOW IT. IF LABSOFT IS NOT IN THE MACHINE, IT WILL LOAD THE PRE-TRIGGER RECORDER BELOW DOS.

TO USE THIS UTILITY, YOU MUST FIRST DIMENSION AN INTEGER ARRAY IN BASIC. THEN YOU CALL THE ROUTINE WITH A LIST OF PARAMETERS. THE FORMAT OF THE CALL STATEMENT IS

CALL STARTING ADDRESS OF ROUTINE, ARRAY NAME, NUMBER OF PRE-TRIGGER SAMPLES, SAMPLING RATE, ANALOG CHANNEL#, TRIGGER VALUE, INTERFACE CARD SLOT# (OPTIONAL)

ALL BUT THE LAST PARAMETER ARE REQUIRED.

PRE-TRIGGER PARAMETER LIST

STARTING ADDRESS OF ROUTINE

THIS IS ALWAYS THE ADDRESS THAT THE PRE-TRIGGER RECORDER WAS LOADED AT.

ARRAY NAME

THIS IS THE NAME OF THE ARRAY THAT THE PRE-TRIGGER RECORDER WILL STORE IT'S DATA IN. IT MUST BE AN INTEGER ARRAY AND IT MUST ALSO HAVE BEEN PREVIOUSLY DIMENSIONED, IF NOT YOU'LL GET AN "OUT OF DATA" ERROR.

NUMBER OF PRE-TRIGGER POINTS

THIS PARAMETER SPECIFIES HOW MANY OF THE ARRAYS ELEMENTS SHOULD BE FILLED BEFORE THE TRIGGER. IF THIS PARAMETER IS GREATER THAN THE SIZE OF THE ARRAY, YOU'LL GET AN ILLEGAL QUANTITY ERROR.

SAMPLING RATE

THIS IS SIMILIAR TO THE SAMPLING RATE PARAMETER IN LABSOFT (RT). IT MUST BE A NUMBER IN THE RANGE 0 - 255. SPECIFYING THE SAMPLE RATE AS 0 WILL CAUSE THE ROUTINE TO ACQUIRE DATA AS FAST AS IT CAN (FREE RUNNING). THE FREE RUNNING SPEED IS APPROX 147 US/SAMPLE (6.8 KHZ). EVERY UNIT ADDED TO THE SAMPLING RATE, ADDS ANOTHER 500 US PER SAMPLE. UNLIKE LABSOFT THIS ROUTINE'S TIMING IS NOT DONE WITH A HARDWARE TIMER AND IT IS NOT NEARLY AS ACCURATE NOR AS STABLE. THE TIMES LISTED BELOW ARE APPROXIMATIONS, AND THEIR WILL BE SLIGHT SAMPLE TO SAMPLE VARIATIONS.

SAMPLE SPEED =

$$147 \text{ US} + (\text{SAMPLING RATE} * 500 \text{ US})$$

SAMPLE RATE TIME/SAMPLE SAMPLES/TIME

0	-----	147 US	-----	6.8 KHZ
1	-----	647 US	-----	1.5 KHZ
10	-----	5.15 MS	-----	194 HZ
100	-----	50.1 MS	-----	20.0 HZ
255	-----	127 MS	-----	7.8 HZ

ANALOG CHANNEL

THIS IS THE A/D CHANNEL# FROM WHICH YOU WISH TO TAKE THE SAMPLES. MUST BE IN THE RANGE 0-15.

TRIGGER VALUE

THIS PARAMETER SPECIFIES WHICH SCHMITT TRIGGER INPUT(S) WILL BE USED TO START THE SAMPLING. THE TRIGGER VALUE SHOULD BE EITHER 1,2,4,OR 8 DEPENDING ON WHICH INPUT YOU CHOOSE TO USE.

INTERFACE CARD SLOT#

THIS IS THE ONLY OPTIONAL PARAMETER FOR THIS ROUTINE. IF THE ISAAC INTERFACE CARD IS IN ANY SLOT OTHER THAN THE NORMAL ONE

(SLOT #3), THEN YOU MUST SPECIFY THE SLOT NUMBER AS THE LAST PARAMETER.

UNLIKE LABSOFT, ALL OF THESE PARAMETERS FOR THIS ROUTINE MUST APPEAR IN THE SPECIFIC ORDER DESCRIBED ABOVE.

PRE-TRIGGER RECORDER!

IF WE WERE TO CONTINUE WITH THE PROGRAM STARTED ABOVE, THE NEXT LINES MIGHT BE

```
240:
250 REM DIMENSION AN AN INTEGER ARRAY
260 DIM ADX(5000)
270:
280 REM EXECUTE THE PRE-TRIGGER RECORDER    PROGRAM
290 CALL ADR,ADX,1000,2,7,1
```

IN THIS EXAMPLE WE FIRST DIMENSION AN ARRAY WITH 5001 ELEMENTS. THEN WE CALL THE PRE-TRIGGER RECORDER UTILITY WITH A LIST OF PARAMETERS. THE FIRST IS THE STARTING ADDRESS OF THE ROUTINE WHICH WAS STORED AS ADR ABOVE. THE NEXT PARAMETER IS THE ARRAY NAME. IN THIS EXAMPLE IT'S ADX.

THEN WE SPECIFY THAT THE FIRST 1000 POINTS SHOULD BE PRE-TRIGGER SAMPLES. NEXT IS THE SAMPLING RATE, WHICH IS SET TO 2. THIS MEANS THAT THE ROUTINE WILL TAKE A NEW SAMPLE APPROXIMATELY EVERY 1 MS. THE A/D CHANNEL#=7 AND THE TRIGGER VALUE=1.

WHAT THIS ROUTINE WILL DO IS CONTINUOUSLY TAKE SAMPLES FROM A/D CHANNEL #7 AND STORE THEM IN THE FIRST SECTION OF THE ARRAY ADX. IF THE FIRST 1000 ELEMENTS OF THE ARRAY GET FILLED BEFORE A TRIGGER HAS BEEN RECEIVED, THEN IT GOES BACK AND STARTS OVER AGAIN FROM THE BEGINNING. AS SOON AS A TRIGGER IS RECEIVED, THE ROUTINE STARTS FILLING THE REST OF THE ARRAY (FROM #1001) WITH MORE A/D SAMPLES.

WHEN THE SAMPLING IS DONE, THE FIRST 1000 POINTS ARE IN THE RIGHT ORDER BUT THEY BEGIN AND END AT THE WRONG PLACE, SO THE ROUTINE TAKES THESE POINTS AND ROTATES THEM UNTILL THEY LINE UP CORRECTLY. THAT IS UNTILL POINT #1000 IS THE LAST POINT THAT WAS TAKEN BEFORE THE TRIGGER. THE PROCESS OF ROTATING THE DATA POINTS CAN BE QUITE TIME CONSUMING IF THERE ARE A LOT OF PRE-TRIGGER SAMPLES. THE TIME IT TAKES TO SHIFT ALL OF THE PRE-TRIGGER POINTS BY ONE ELEMENT = # OF ELEMENTS * 50 US. IF THERE ARE 1000 POINTS, THEN THE SAMPLES MIGHT HAVE TO BE MOVED AS MANY AS 999 TIMES, SO THE TIME INVOLVED COULD BE AS MUCH AS 49 SECONDS. THE CHART BELOW IS AN APPROXIMATION OF THE WORST CASE TIMING FOR VARIOUS NUMBERS OF POINTS.

NUMBER OF PRE-TRIGGER POINTS	WORST CASE TIMES
---------------------------------	------------------

10 POINTS	----- 50 MS
100 POINTS	----- 500 MS
1000 POINTS	----- 50 SEC
2000 POINTS	----- 200 SEC

3000 POINTS ----- 450 SEC
5000 POINTS ----- 1250 SEC
10000 POINTS ----- 83.3 MIN

THESE TIMES ARE FOR JUSTIFYING THE ARRAY, NOT TAKING THE SAMPLES. THE NUMBER OF POINTS IN THE TABLE ABOVE REFERS TO THE NUMBER OF PRE-TRIGGER POINTS NOT THE TOTAL SIZE OF THE ARRAY. THESE TIMES ARE WORST CASE, THE AVERAGE TIME IS HALF THAT LISTED.

IF YOU RECEIVE A TRIGGER BEFORE FILLING THE PRE-TRIGGER SAMPLE SECTION OF THE ARRAY, THE ARRAY WILL STILL BE JUSTIFIED AND THE FIRST SAMPLES WILL BE FILLED WITH ZEROS. FOR EXAMPLE IF YOU DIMENSION AN ARRAY FOR 10 VALUES AND YOU SPECIFY THAT FIVE OF THEM SHOULD BE PRE-TRIGGER SAMPLES AND THEN WHEN YOU RUN THE ROUTINE, YOU GET A TRIGGER AFTER THE 2ND SAMPLE THE RESULTS MIGHT LOOK LIKE THIS

AD%(0) = 0	AD%(5)=3456
AD%(1) = 0	AD%(6)=3460
AD%(2) = 0	AD%(7)=3462
AD%(3) = 3042	AD%(8)=3467
AD%(4) = 3048	AD%(9)=3472

-- APPLE SLICER --

APPLE SLICER IS A UTILITY PROGRAM WHICH ALLOWS YOUR APPLESOFT PROGRAMS TO USE ALL AVAILABLE MEMORY IN THE MACHINE WHEN HIGH RESOLUTION GRAPHICS ARE USED.

SINCE THE HIRES PAGES ARE LOCATED ON THE 8K AND 16K BOUNDARIES, AND SINCE THIS AREA IS NORMALLY USED FOR BASIC PROGRAMS AND VARIABLES, USING HIRES GRAPHICS USUALLY REQUIRES YOU TO LIMIT THE SIZE OF YOUR PROGRAM TO 16K OR LESS.

OFTEN YOU HAVE TO GO THROUGH OTHER OPERATIONS (HIMEM, LOMEM, ETC) TO GET EVEN THE 16K.

THIS UTILITY SOLVES THE PROBLEM BY SPLITTING ANY BASIC PROGRAM INTO TWO PARTS:

THE FIRST 8K OF THE PROGRAM IS LEFT IN ITS NORMAL POSITION IN FRONT OF THE HIRES PAGE #1 (8K - 16K).

THE SECOND PART OF THE PROGRAM IS MOVED UP IN MEMORY TO JUST ABOVE ONE OF THE HIRES SCREENS.

SLICER.OBJ WILL MOVE IT PAST HIRES SCREEN #1

SLICEBOTH.OBJ WILL MOVE IT PAST HIRES SCREEN #2

TO USE THIS UTILITY, A PROGRAM MUST MEET THE FOLLOWING REQUIREMENTS:

1. IT SHOULD USE HIGH RESOLUTION GRAPHICS. NO ADVANTAGE IS GAINED FOR PROGRAMS THAT DON'T.
2. IT MUST BE LONGER THAN 8K IN LENGTH
3. THE LENGTH OF THE PROGRAM PLUS 8K FOR THE HIRES SCREEN MUST NOT EXCEED THE CURRENT VALUE FOR HIMEM.

TO USE APPLE SLICER ON A PROGRAM, THE APPROPRIATE SLICER PROGRAM (EITHER SLICER.OBJ OR SLICEBOTH.OBJ) MUST BE ON A DISK IN THE DRIVE AT RUN TIME.

THE FIRST THING THE PROGRAM TO BE SPLIT MUST DO (BEFORE ANY VARIABLES ARE REFERENCED !) IS TO BRUN THE SLICER ROUTINE.

REMEMBER THIS MUST BE DONE BEFORE ANY VARIABLES ARE REFERENCED. APPLE SLICER WILL SPLIT THE BASIC PROGRAM AND CLEAR ALL VARIABLES.

BOTH OF THE SLICER PROGRAMS ARE RELOCATABLE. THE BEST PLACE TO

RUN THEM IS UNDER HIMEM. IF YOU PUT THE SLICER THERE, IT WON'T BOTHER DOS, LABSOFT, OR ANY OTHER UTILITY PROGRAMS THAT MIGHT ALREADY HAVE BEEN LOADED.

TO DO THIS USE THE FOLLOWING LINES AT THE VERY BEGINNING OF YOUR BASIC PROGRAM.

```
100 X=PEEK(115)+PEEK(116)*256
```

```
110 ADR = X - 340
```

```
120 PRINT CHR$(4);"BRUN SLICER.OBJ,A"; ADR
```

*** WARNINGS ***

APPLE SLICER MUST BE RUN AS THE FIRST LINES OF AN EXECUTING PROGRAM. DO NOT ATTEMPT TO RUN IT IN THE IMMEDIATE EXECUTION MODE.

ONCE A PROGRAM IS SPLIT, DON'T TRY TO EDIT THE PROGRAM.

ALSO DO NOT ATTEMPT TO SAVE OR LOAD PROGRAMS THAT HAVE BEEN SPLIT.

NEVER ATTEMPT TO SPLIT A PROGRAM MORE THAN ONCE.

DOING ANY OF THE ABOVE WILL RE-LINK THE BASIC TEXT AND THE SECOND HALF OF THE PROGRAM IN MEMORY WILL BE LOST.

APPENDIX X: THE EXPANSION COMMANDS

The hardware portion of the ISAAC/LabSoft/Apple system is, as you may have noted, designed to be easily expanded. To complement the expandable hardware, LabSoft comes equipped with two expansion commands. These commands are unsupported in the 91A ISAAC as it is currently shipped, and you need not concern yourself with them unless you are planning to incorporate devices in your system which the other LabSoft commands are unable to access efficiently, or at all. In essence, these two commands are primitive read and write orders, not tailored to the specific needs of Analog, Binary, Counter, or Timer/Clock devices.

X.1 & RDEV & @RDEV

```
& RDEV ,(D#) = aexpr ,(W#) = aexpr ,(TV) = avar
& @RDEV ,(D#) = aexpr ,(W#) = aexpr
         ,(AV) = numaryname
```

This command (simple or matrix) reads a value (or series of values) from the specified device number and the specified word number into the specified target variable. This is LabSoft's universal input command and with it, you can communicate from a BASIC program on a primitive level with any present or future ISAAC hardware. This does not work particularly well with the presently defined hardware; you will find other commands more appropriate. However, this command could be very useful for the BASIC programmer who wants to access custom hardware connected to ISAAC, or for the programmer who wants to try something different with the present hardware. This could also be advantageous for the assembly language programmer who would like to experiment with "direct hardware access."

NOTE In addition to the usual required input parameter (TV), this command also REQUIRES specification of a device number (D#) parameter and a word number (W#) parameter. (W#), you may have noticed, is only used with these two expansion commands.

The following parameters are optional with & RDEV.

Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
Analog Graph Flag	(GA)	"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with & @RDEV.

Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
Analog Graph Flag	(GA)	"XOR" Mask	(XM) =
		Compare Value	(CV) =
		No. of SWeeps	(SW) =
		Sampling RaTe	(RT) =

X.2 & WRDEV & @WRDEV

& WRDEV (D#) = aexpr, (W#) = aexpr, (DV) = aexpr
& @WRDEV, (D#) = aexpr, (W#) = aexpr, (AV) = aexpr

These commands are the output complement to & RDEV and & @RDEV above. The principal difference in their syntactic requirements is that, being output commands, they require a Data Value (DV) parameter instead of a Target Value (TV).

NOTE In addition to the usual required output parameter (DV), this command also REQUIRES specification of a device number (D#) parameter and a word number (W#) parameter. (W#), you may have noticed, is only used with these two expansion commands.

The following parameters are optional with & WRDEV.

Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
Analog Graph Flag	(GA)	"XOR" Mask	(XM) =
		Compare Value	(CV) =

The following parameters are optional with & @WRDEV.

Binary Graph Flag	(GB)	FUnction	(FU) =
Print Raw Value Flag	(PR)	"AND" Mask	(AM) =
Analog Graph Flag	(GA)	"XOR" Mask	(XM) =
		Compare Value	(CV) =
Sampling RaTe	(RT)	No. of SWeeps	(SW) =

APPENDIX Z: INDEX

Where a LabSoft command or parameter appears in this index, the page numbers referred to indicate where the command is initially introduced as well as where it is first used in a demonstration program or other context.

- & AIN ... 56
- & @AIN ... 56
- Alternating Graph ... 15,16,28-30
- & ALTSET ... 21
- (AM) ... 42
- & ANAFMT ... 60,61
- & AOUT ... 58,59
- & @AOUT ... 58,59
- Array Variables ... 35,36,61
- & ARRAYCLR ... 10,58
- & ASUM ... 57,58
- & @ASUM ... 57 ... 58
- (AV) ... 35
- & BCDIN ... 70
- & @BCDIN ... 70
- & BCDOUT ... 72
- & @BCDOUT ... 72
- & BEEP ... 9
- & BEEP ON ... 9
- & BEEP STOP ... 9
- & BIN ... 64
- & @BIN ... 64
- Binary Graph ... 17,30,31
- & BINEMT ... 25,31
- & BOUT ... 67
- & @BOUT ... 67
- & BPOLL ... 73
- & @BPOLL ... 73
- & BUZZ ... 9
- & BUZZ ON ... 9
- & BUZZ STOP ... 9
- & CLRCOUNTER ... 80
- & CLRTIMER ... 88
- & CNTFMT ... 80,81
- & COUNTERIN ... 82
- (CV) ... 48,49
- (C#) ... 38
- & DAY TO ... 89
- (DV) ... 34,35

(D#) ... 37, Appendix X
 EQ% ... 48,49, Appendix B
 & ERRPTCH ... 11
 (FU) ... 39-41
 & FINH ... 85
 & @FINH ... 85
 & FINL ... 83
 & @FINL ... 83
 & FMTDELT ... 12
 Frequency Counting ... 83,84
 & FULLSCREEN ... 19
 (GA) ... 45,46
 (GB) ... 46,47
 GT% ... 48,49 Appendix B
 Handshaking
 Binary Input ... 65,66
 Binary Output ... 67,68
 & HIRES1 ... 19
 & HIRES2 ... 19
 I/O Sequences ... 52,53
 & LABEL ... 25-27
 LabSoft Master Disk ... 5,6
 LabSoft Utilities Disk ... 8, Appendix U
 & LOOK FOR...THEN ... 51,52,91-95
 LT% ... 48,49, Appendix B
 Matrix Operations ... 50,51,65,94
 MASKED% ... 52,53, Appendix B
 & MIXSCREEN ... 20
 & NXTBIN ... 24,31
 & NXTPLT ... 24,29,30
 & OFFERR ... 11
 OUT ... 41,53
 & OUTLINE ... 21
 & PAUSE ... 13
 & PLTFMT ... 22,23,28,30
 (PR) ... 46,47
 RAW% ... 39,40, Appendix B
 & RDEV ... Appendix X
 & @RDEV ... Appendix X
 Reserved Variables ... 52,53, Appendix B
 & RETRCE ... 22,29
 (RT) ... 47,48
 Schmitt Trigger ... 75,76
 Scrolling Graph ... 13,14
 & SCROLLSET ... 20
 Signal Averaging ... 57,58
 & SLOT# ... 11
 Syntax Conventions ... 3,4
 (SW) ... 48
 & TIME TO ... 89

& TIMERIN ... 88
(TH) ... 37,91-93
Triggered Operations ... 51
& TRIGIN ... 75
& @TRIGIN ... 75
& TRIGPOLL ... 77
& @TRIGPOLL ... 77
(TV) ... 34
& WRDEV ... Appendix X
& @WRDEV ... Appendix X
(W#) ... 48, Appendix X
(XM) ... 42

SEE ALSO "ISAAC 91A SYSTEM INDEX", LOCATED IN THE
APPENDICES TO THE USER'S GUIDE.



NOTES



